TUGAS AKHIR – TM 141585

PENENTUAN PREDIKSI KOORDINAT 3 DIMENSI DENGAN METODE *EXPONENTIAL SMOOTHING* UNTUK TARGET TUNGGAT PADA SISTEM PELONTAR PELURU *AUTO-TRACKING*

Valya Ika Dhanie
NRP. 2112 100 036

Dosen Pembimbing
Arif Wahjudi, St., Mt., Phd.

JURUSAN TEKNIK MESIN
Fakultas Teknologi Industri
Institut Teknologi Sepuluh Nopember
Surabaya 2016

FINAL PROJECT – TM 141585

DETERMINING 3 DIMENSION COORDINATE PREDICTION USING EXPONENTIAL SMOOTHING METHOD FOR ONE TARGET ON AUTO-TRACKING SISTEM BULLET LAUNCHER

Valya Ika Dhanie
NRP. 2121 100 036

Academic Advisor
Arif Wahjudi, St., Mt., Phd.

DEPARTMENT MECHANICAL ENGINEERING
Faculty of Industrial Technology
Sepuluh Nopember Institute of Technology
Surabaya 2016

LEMBAR PENGESAHAN

PENENTUAN PREDIKSI KOORDINAT 3 DIMENSI DENGAN METODE *EXPONENTIAL SMOOTHING* UNTUK TARGETTUNGGAL PADA SISTEM PELONTAR PELURU *AUTO-TRACKING*

TUGAS AKHIR

Diajukan untuk Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Teknik
pada
Program Studi S-1 Jurusan Teknik Mesin
Fakultas Teknologi Industri
Institut Teknologi Sepuluh Nopember

Oleh:

VALYA IKA DHANIE
Nrp. 2112 100 036

Disetujui oleh Tim Penguji Tugas Akhir :

1. Arif Wahjudi, ST, MT., PhD ..... (Pembimbing)
   NIP. 197303222001121001
2. Dinny Harnany, ST, MSc. ..... (Penguji I)
   NIP. 2100201405001
3. Dr. Eng Unggul Wasiwitono, ST, M.Eng. Sc. ..... (Penguji II)
   NIP. 197805102001121001
4. Latifah Nurahmi, ST, MSc, Ph.D ..... (Penguji III)
   NIP. 210000011

SURABAYA
Juli 2016

PENENTUAN PREDIKSI KOORDINAT 3 DIMENSI DENGAN METODE *EXPONENTIAL SMOOTHING* UNTUK TARGET TUNGGAL PADA SISTEM PELONTAR PELURU *AUTO-TRACKING*

Nama Mahasiswa : Valya Ika Dhanie
NRP : 2112 100 036
Jurusan : Teknik Mesin
Dosen Pembimbing : Arif Wahjudi, St., Mt., Phd.

Abstrak

Wilayah negara Indonesia terbentang sepanjang 3.977 mil, dengan luas lautan 3.257.483 km² dan luas daratan 1.922.570 km². Tidak jarang daerah-daerah perbatasan Indonesia masih rawan dan mudah ditembus oleh pihak asing. Untuk meminimalkan jumlah SDM (Sumber Daya Manusia), mulai dikembangkan sistem persenjataan yang memanfaatkan mesin yang diharapkan lebih cepat dan lebih akurat sebagai ganti militer manusia. Mesin tersebut bisa dikembangkan dengan memanfaatkan teknologi *non contact measurement* dan teknologi pendektesian objek yang dapat mendeteksi posisi suatu objek seperti proses pengolahan citra. Sistem pelontar peluru *auto-tracking* sendiri bertujuan untuk mendeteksi suatu peluru lalu menembaknya di udara sebelum peluru tersebut mengenai target. Dari proses memasukkan koordinat sampai dengan penembakan peluru pasti ada jeda waktu, dan dalam selang waktu itu posisi peluru sudah berpindah, untuk itu kita perlu memprediksi posisi peluru selanjutnya.

Sebelum menerapkan metode *exponential smoothing* untuk prediksi 3 dimensi, akan dibuat dulu secara 2 dimensi (x dan y) dengan menggunakan input 10 video untuk merepresentasikan *real time*, hasil dari prediksi ini akan dihitung nilai kesalahannya menggunakan metode *root mean square* lalu hasil *error* ini dibandingkan dengan *error* yang didapatkan mengunakan metode filter kalman. Jika selisih *error* yang didapat sudah kurang dari 0,1 maka hasil metode *exponential smoothing* dianggap sudah layak

dengan nilai alfa tertentu dan dikembangkan menjadi prediksi 3 dimensi (x, y, dan z).

Nilai alfa didapat sebesar 0.4995 dan dari 10 video 6 diantaranya menunjukkan bahwa metode *exponential smoothing* mempunyai E_{RMS} lebih sedikit dibanding filter kalman. Hasil nilai error terbesar pada metode filter kalman untuk koordinat x adalah 8.55cm dan untuk koordinat y sebesar 18.34cm, sedangkan hasil error terbesar metode *exponential smoothing* untuk koordinat x adalah 11.126cm dan untuk koordinat y sebesar 12.77cm.

Kata kunci: prediksi 3D, exponential smoothing, filter kalman, image processing

DETERMINING 3 DIMENSION COORDINATE PREDICTION USING EXPONENTIALSMOOTHING METHOD FOR ONE TARGET ON AUTO-TRACKING SISTEM BULLET LAUNCHER

Student Name : Valya Ika Dhanie
NRP : 2112 100 036
Department : Mechanical Engineering
Academic Advisor : Arif Wahjudi, St., Mt., Phd.

Abstract

Indonesia has length up to 3.977 mil, covered with 3.257.483 km² ocean area and 1.922.570 km² land area. So the boundary of this country is also large. For optimization human resource we start developing machine weapon system, aim that they will move faster and more accurate compare to human. That machine could using non contact measurement and detection program like image processing that can detect position of some object. This auto-tracking system bullet launcher have purpose for detect some missile and hunt it down before those missile get in the target. The proses have some delay time from inputing the coordinate until we shooting the bullet, in those time the position of missile is keep changing, so we have and need to predict the possition of missile if we don't want to miss it.

Before we use exponential smoothing for 3 dimention prediction, we create the 2 dimentional first, using 10 video for the input as substitute for real time, from this process we can calculate the error or how much the different between the prediction and the real time position using root mean square error method, we will compare that value using exponential smoothing method and using kalman filter method. If the different between those two less than 0,1 we can accept the alfa, and use that alfa for 3 dimentional prediction.

We got that alfa is 0.4995, 6 of 10 video shows that exponential smoothing have less E_{RMS} than kalman filter. The biggest error for kalman filter in x coordinate is 8.55cm, and for y coordinate is 18.34cm. While the biggest error using exponential method for x coordinate is 11.126cm and 12.77cm for y coordinate.

Key Word: prediksi 3D, exponential smoothing, kalman filter, image processing

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

(Halaman sengaja dikosongkan)

DAFTAR LAMPIRAN

BAB I
PENDAHULUAN

1.1 Latar Belakang

Negara Indonesia merupakan negara yang berbentuk kepulauan terbesar di dunia dengan jumlah pulau sebanyak 13.466 pulau serta terletak di antara samudera Pasifik dan Indo-Australia serta berada di antara benua Asia dan Australia. Wilayah negara Indonesia terbentang sepanjang 3.977 mil, dengan luas lautan 3.257.483 km² dan luas daratan 1.922.570 km². Tidak jarang daerah-daerah perbatasan Indonesia masih rawan dan mudah ditembus oleh pihak asing.

Untuk meminimalkan jumlah sumber daya manusia, mulai dikembangkan sistem persenjataan dengan memanfaatkan mesin yang diharapkan lebih cepat dan lebih akurat sebagai ganti militer manusia. Mesin tersebut bisa dikembangkan dengan memanfaatkan teknologi *non contact measurement* dan teknologi pendektesian objek yang dapat mendeteksi posisi suatu objek.

Adapun yang sudah dikembangkan hingga saat ini adalah mengukur jarak menggunakan gelombang ultrasonik. Prinsip kerjanya adalah mendeteksi jarak objek dengan cara memancarkan gelombang ultrasonik selama beberapa waktu kemudian mendeteksi pantulannya. Dengan cepat rambat udara diketahui maka kecepatan gelombang didapat dan jarak dapat dihitung [1].

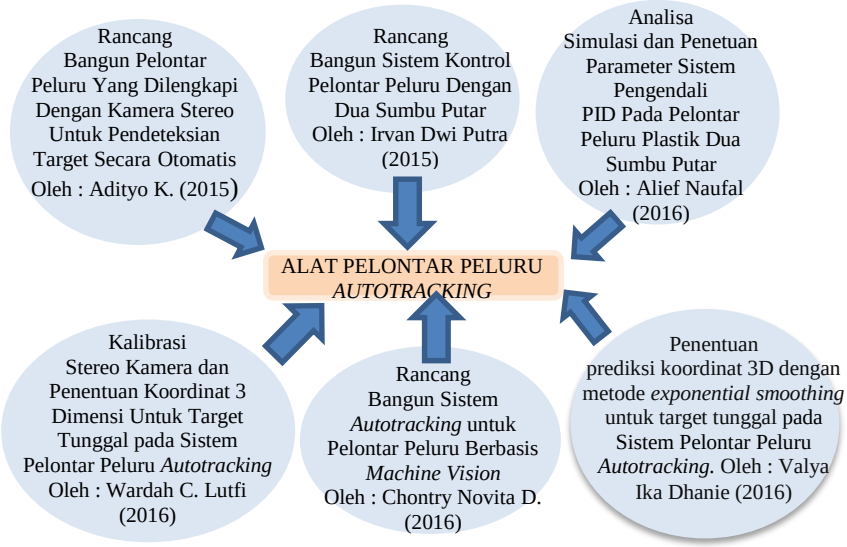
Saat ini sudah banyak dikembangkan senjata tanpa awak (*autonomus gun*) yang menggunakan kamera untuk sensor inputannya dimana senjata ini bisa mendeteksi suatu objek namun hanya menggunakan 1 kamera sehingga tidak bisa memperkirakan posisi 3 dimensi target [2]. Adapun pendeteksian menggunakan kamera lebih sering dikenal sebagai pengolahan citra (*image processing*). Pengolahan citra adalah teknik mengolah citra yang mentransformasikan

citra masukan menjadi citra lain agar citra keluaran memiliki kualitas yang lebih baik dibandingkan kualitas citra masukan. Pengolahan citra sangat bermanfaat, diantaranya adalah untuk meningkatkan kualitas citra, menghilangkan cacat pada citra, mengidentifikasi objek, dan melakukan penggabungan dengan bagian citra yang lain.

Untuk menghasilkan suatu alat pelontar peluru *auto-tracking* yang mampu mendeteksi dan menembak objek secara otomatis dengan sensor kamera ini terdiri dari berbagai pokok pembahasan (gambar 1.1), yaitu : rancang bangun alat pelontar peluru, rancang bangun sistem kontrol dari alat pelontar peluru tersebut, penentuan sistem pengendali PID pada alat pelontar peluru, penambahan sensor kamera yang dapat menentukan koordinat 3 dimensi dari objek, dan rancang bangun sistem alat pelontar peluru berbasis *machine vision,* serta penentuan prediksi koordinat 3 dimensi dari objek.

Dalam tugas akhir berjudul kalibrasi stereo kamera dan penentuan koordinat 3 dimensi utuk target tunggal pada sistem pelontar peluru *auto-tracking* oleh saudari Wardah C. Lutfi (2016) sudah bisa mendeteksi benda yang diam dan menentukan posisinya terhadap kamera, namun sistem pelontar peluru *auto-tracking* sendiri bertujuan untuk mendeteksi suatu peluru lalu menembaknya di udara sebelum peluru tersebut mengenai target. Dari proses input koordinat sampai dengan penembakan peluru pasti ada jeda waktu, dan dalam selang waktu itu posisi peluru yang kita deteksi sudah berpindah, sehingga jika kita memasukkan koordinat secara *real-time* pada senjata akan terlambat untuk mengejar pergerakan peluru yang kita deteksi. Untuk menghindari hal tersebut kita perlu memprediksi posisi selanjutnya dari peluru

yang kita deteksi, yang nantinya akan berperan sebagai masukkan dari sistem pelontar peluru *auto-tracking.*



Gambar 1.1 Pemetaan judul dalam tugas akhir sistem pelontar peluru *autotracking.*

1.2 Rumusan Masalah

Adapun dalam tugas akhir ini rumusan masalah yang akan dibahas adalah :

1. Bagaimana mendapatkan prediksi koordinat 3 dimensi menggunakan *exponential smoothing?*
2. Bagaimana hasil perbandingan prediksi koordinat 2 dimensi menggunakan metode *exponential smoothing* dan filter kalman?

1.3 Batasan Masalah

Batasan masalah pada proposal tugas akhir ini adalah sebagai berikut :

1. Kamera yang digunakan posisinya tidak bergerak dengan tipe *CMOS* (*Complementary Metal-Oxide Semiconductor*).
2. Menggunakan 2 kamera *(stereo vision).*
3. Hanya bisa mendetekteksi objek tunggal.
4. Objek yang dipakai adalah bola berdiameter 5 cm dan berwarna kuning.
5. Jarak minimal pendeteksian yaitu 80 cm dan jarak maksimal sejauh 5 meter.
6. Menggunakan sistem pencahayaan *indoor*.
7. Kamera lurus menghadap ke depan

1.4 Tujuan

Tujuan dari tugas akhir ini adalah :

1. Untuk mendapatkan prediksi koordinat 3 dimensi menggunakan *exponential smoothing*.
2. Mengetahui hasil perbandingan prediksi koordinat 2 dimensi menggunakan metode *exponential smoothing* dan filter kalman?

1.5 Manfaat Penelitian

Penelitian ini diharapkan dapat memberikan manfaat sebagai berikut:

1. Dapat menambah wawasan terkait *image processing*, terlebih untuk pendeteksian menggunakan warna.
2. Sebagai bahan referensi bagi penelitian sejenisnya dalam rangka pengembangan pengetahuan tentang *image processing*.
3. Dapat digunakan sebagai alat ukur jarak jauh.

BAB II
TINJAUAN PUSTAKA

2.1 Tinjauan Pustaka

Saat ini kemajuan teknologi untuk mendeteksi serta mengestimasi pergerakan suatu benda sudah banyak dikembangkan. Hal ini tentu akan mempermudah manusia seperti sistem kunci di *smart phone* yang menggunakan deteksi sidik jari ataupun pemindai wajah. Paper-paper yang ada juga sudah mulai membahas tentang estimasi gerak selanjutnya dari suatu benda, jadi tidak hanya mendeteksi saja.

Dengan menggunakan 1 kamera kita bisa mendapatkan kecepatan suatu benda bergerak yang kita deteksi yaitu dengan cara merekam video secara *real time*, kemudian dari video tersebut dipecah menjadi banyak *frame*, lalu di bandingkan dari 1 *frame* ke *frame* yang lain nilai piksel yang berubah (menandakan adanya pergerakan) lalu mencari nilai tengah benda disetiap *frame* dan kemudian menjadi sebuah lintasan. Lalu akan di transformasikan ke jarak sebenarnya dengan nilai intrinsik kamera yang didapat [3]. Kelemahannya adalah masih menggunakan 1 kamera sehingga yang tertangkap hanya dalam posisi x dan y saja.

Program untuk estimasi posisi sudah mulai dikembangkan dengan menggunkan filter kalman. Dari sensor *accelerator* didapatkan percepatan benda, lalu di *integral* sebanyak dua kali untuk mendapatkan posisi, lalu posisi ini diestimasi menggunakan filter kalman dengan pemberian nilai *noise* yang berbeda dan dilihat pengaruhnya [4]. Namun dalam uji cobanya hanya untuk koordinat x saja.

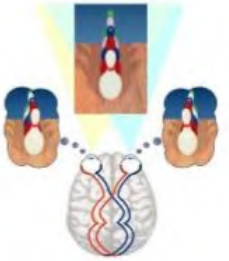
Image processing bisa juga diterapkan dalam dunia olahraga, pada paper yang saya baca *image processing* (pengolahan citra) digunakan untuk menentukan gerakan bola dan pemain selanjutnya dengan cara memisahkan bagian gambar yang tidak bergerak (statik) dan dijadikan sebagai *background* dengan bagian gambar yang bergerak. Ukuran bola sebelumnya sudah diketahui, sehingga dengan perbandingan akan didapatkan koordinat x, y, dan z dari bola [5].

2.2 Dasar Teori

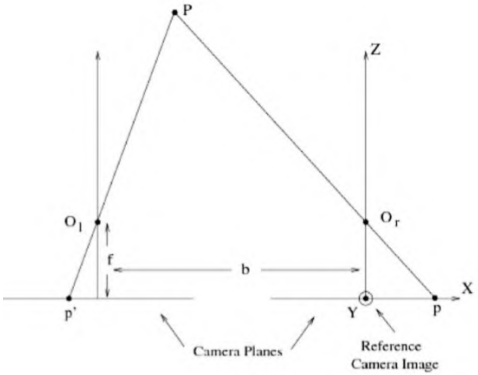
2.2.1 Kamera Stereo (*Stereo Vision*)

Binocular vision didefinisikan sebagai sebuah gambar yang terlihat dari 2 bola mata, dimana data yang didapat dari masing-masing mata dan saling tumpang tindih di sebagian tempat. Bagian yang saling tumpang tindih tadi digunakan *biological vision* untuk mendapatkan kedalaman. *Stereoscopic vision* menggunakan *binocular vision* untuk mendapatkan struktur 3 dimensi dalam dunia nyata. Disparitas binokular adalah perbedaan tempat diletakkannya sebuah objek menurut mata kita, dimana masing-masing mempunyai sudut pandang yang berbeda tergantung mata mana yang digunakan. Sebuah sistem *stereo vision* terdiri dari dua atau lebih kamera, digunakan untuk mendapatkan kedalaman *scene* 3D yang dilihat dari berbagai sudut pandang, sebagai model dari *binocular vision* pada manusia. Gambar 2.1 menunjukkan bagaimana *stereo vision* pada manusia [6].

Sistem stereo paling sederhana terdiri dari 2 kamera dimana pusat optiknya (O_l dan O_r) terpisah sejauh b. Asumsikan 2 kamera memiliki karakteristik optik yang identik dan sensor optik mereka koplanar (sebidang) seperti yang ditunjukkan pada gambar 2.2. sebuah *stereo system* bisa menangkap 2 gambar (l_l dan l_r) secara bersamaan, pada contoh ini *reference image* berada pada pusat gambar sebelah kanan.



Gambar 2.1 : Contoh *binocular vision* pada manusia [6].



Gambar 2.2 Sistem stereo terdiri dari 2 kamera [6]

Sebuah titik P = (Xp, Yp, Zp) dalam gambar 2.2 diproyeksikan kedalam 2 lokasi (p=(x,y) dan p'=(x',y)). Dengan mengetahui parameter intrinsik dari *system stereo* memungkinkan kita untuk merekontruksi ulang struktur 3 dimensi dari piksel yang disparitasnya diketahui. Kedalaman (*depth*) dihitung dengan rumus [3].

$$Zp = \frac{fb}{d} \quad \text{.....(1)}$$

$$Xp = \frac{xZ}{f} \quad \text{.....(2)}$$

$$Yp = -\frac{yZ}{f} \quad \text{.....(3)}$$

Dimana f adalah *focal length* dari kamera (didapatkan dari kalibrasi kamera) dan d adalah disparitas yaitu perbedaan nilai piksel antara gambar kanan dan kiri. (d=|x-x'|). Sehingga untuk mendapatkan kedalaman (jarak antara kamera dengan objek) kita membutuhkan nilai *focal length*, jarak antara 2 kamera, dan disparitas. Sedangkan untuk menghitung X dan Y sebagai posisi 3 dimensi dari titik P kita membutuhkan nilai piksel dari

reference image (x dan y), nilai kedalaman, dan juga *focal length* dari kamera [3].

2.2.2 *Computer Vision*

Dua sampai tiga dekade terakhir pengembangan komputer berjalan sangat cepat, bersamaan dengan berkembangnya performa kamera, keduanya telah membuka daerah penelitian baru bernama *computer vision*. *Computer vision* melengkapi komputer dengan alat perasa (*sensing device*), yang sebagian besar berupa optikal sehingga membuat komputer bisa "melihat" lingkungan, mengekstrak informasi, dan kemudian menginterpretasikannya, dan dalam beberapa kasus membuat robot bereaksi terhadapnya [7].

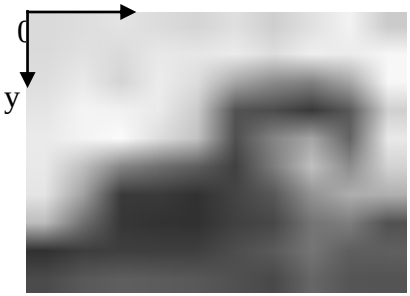
Computer vision memiliki banyak pendukung yang dibutuhkan untuk bisa berfungsi secara baik, ini dilakukan agar *computer vision* mampu menangkap informasi secara maksimal. Fungsi-fungsi tersebut diantaranya yaitu: *image acquisition*, *image processing*, *image analysis*, *image understanding* [8].

2.2.3 Citra Digital

Citra digital dapat didefinisikan sebagai fungsi dua variabel, f(x,y), dimana x dan y adalah koordinat spasial dan nilai f(x,y) adalah intensitas citra pada koordinat tersebut. Teknologi dasar untuk menciptakan dan menampilkan warna pada citra digital berdasarkan pada penelitian bahwa sebuah warna merupakan kombinasi dari tiga warna dasar, yaitu merah, hijau, dan biru (*Red, Green, Blue* - RGB). Dari panjang dan tinggi citra digital tersebut. Panjang citra adalah jumlah kolom pixel pada citra, sedangkan tinggi adalah jumlah baris pada citra [9].

Salah satu contoh bentuk citra digital adalah citra monokrom atau citra hitam putih yang merupakan citra satu kanal, dimana citra f(x,y) merupakan fungsi tingkat keabuan dari hitam ke putih; x menyatakan variabel baris dan y menyatakan

variabel kolom. Gambar 2.3 menunjukkan contoh bentuk citra monokrom dan bentuk matriks penyusunnya [10].



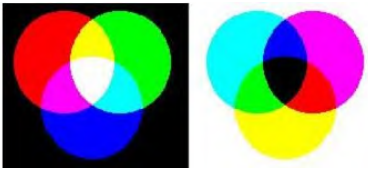
223	228	230	210	215	222	211	224	237	215
204	230	212	243	225	241	223	244	239	249
227	241	208	241	223	172	140	124	177	242
231	242	239	223	212	66	74	48	89	199
231	252	253	225	198	70	140	175	110	225
218	197	122	122	89	67	121	189	142	216
240	170	61	63	52	81	91	145	175	182
175	114	62	37	46	54	70	122	128	81
46	63	64	59	64	71	89	118	98	95
83	96	94	90	91	81	71	78	81	79

(a) (b)

Gambar 2.3 Contoh bentuk citra monokrom dan bentuk matriks penyusunnya [10]

2.2.4 Citra Berwarna

Citra berwarna adalah citra yang direpresentasikan dalam beberapa kanal (*channel*) yang menyatakan komponen-komponen warna penyusunnya. Banyaknya kanal yang digunakan bergantung padamodel warna yang digunakan pada citra tersebut. Sistem grafik citra warna memiliki satu set nilai tersusun yang menyatakan berbagai tingkat warna. Setiap piksel pada citra warna mewakili warna yang merupakan kombinasi dari tiga warna dasar *Red Green Blue* (RGB).

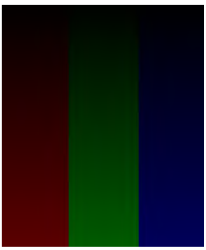


Gambar 2.4 Warna RGB [11]

Citra warna 8 bit adalah citra yang setiap piksel, jumlah warna maksimum yang dapat digunakan adalah 256 warna. Setiap titik (piksel) pada citra warna mewakili warna yang

merupakan kombinasi dari tiga warna dasar yaitu merah, hijau, dan biru yang biasa disebut citra RGB (*Red, Green, Blue*) seperti pada gambar 2.4 di atas.

Citra warna 16 bit atau citra *highcolor* (gambar 2.5) adalah citra yang setiap pikselnya diwakili dengan 2 *byte memory* (16 bit). Warna 16 bit memiliki 65.536 warna. Dalam formasi bitnya, nilai merah dan biru mengambil tempat di 5 bit di kanan dan kiri. Komponen hijau memiliki 5 bit ditambah 1 bit ekstra [11].



Gambar 2.5 Citra warna 16 bit highcolor [11]

Untuk merubah suatu citra dengan warna penuh (RGB) menjadi suatu citra *grayscale* (gambar keabuan), terdapat metode yang umum digunakan, yaitu dengan rumus:

$$\frac{R+G+B}{3} \quad \text{.....(4)}$$

dimana : R : Unsur warna merah
G : Unsur warna hijau
B : Unsur warna biru

2.2.5 HSV

Pada pengolahan suatu warna citra terdapat bermacam – macam model warna. Model dengan menggunakan warna dasar merah, hijau, biru atau yang biasa disebut RGB (*red green blue*) merupakan model yang paling sering digunakan, salah satunya sperti yang digunakan pada layar monitor dan televisi. Pada

model ini 3 buah komponen warna tersebut untuk merepresentasikan suatu citra. Selain model RGB terdapat juga model HSV dimana model ini terdapat 3 komponen yaitu, *hue*, *saturation*, dan *value*.

1. *Hue* merupakan suatu ukuran panjang gelombang yang terdapat pada warna dominan yang diterima oleh penglihatan. *Hue* menyatakan warna yang sebenarnya, seperti merah, violet, kuning dan digunakan untuk menentukan kemerahan (*redness*), kehijauan (*greenness*) dan sebagainya.
2. *Saturation* atau *chroma* adalah kemurnian atau kekuatan warna, ukuran banyaknya cahaya putih yang bercampur pada *hue*.
3. *Value* adalah nilai kecerahan dari warna. Nilainya mulai dari 0 – 100 %. Apabila nilainya 0 maka warna yang dihasilkan menjadi hitam, semakin besar nilai maka semakin cerah dan muncul variasi – variasi baru dari warna tersebut.

Untuk mengubah citra HSV menjadi citra RGB bisa digunakan persamaan seperti berikut ini:

$$V = \max(r, g, b) \quad \text{.....(5)}$$

$$S = \begin{cases} 0, & v = 0 \\ 1 - \frac{\min(r,g,b)}{v}, & v > 0 \end{cases} \quad \text{.....(6)}$$

$$H = \begin{cases} 0, jika\ S = 0 \\ \frac{60*(g-b)}{s*v}, jika\ v = r \\ 60*\left[2 + \frac{b-r}{s*v}\right], jika\ v = g \\ 60*\left[4 + \frac{r-g}{s*v}\right], jika\ v = b \end{cases} \quad \text{.....(7)}$$

Dimana *value* dipilih dari nilai *red, green, blue*, yang mana diantara ketiganya yang mempunyai nilai paling besar. *Saturation* bernilai 0 jika *value* 0, jika *value* bernilai lebih dari 0 maka akan digunakan rumus nomer 3, dimana *min (r,g,b)* adalah nilai yang paling kecil diantara nilai *red, green, dan blue*.

Jika *saturation* S=0, maka *hue* tidak terdefinisi atau dengan kata lain tidak memiliki *hue* berarti *monochrome*. *Hue* (H) lalu dikonversi menjadi derajat/*degrees* dengan cara mengalikan dengan 60 sehingga menghasilkan HSV dengan S dan V antara 0 dan 1 dan H antara 0 – 360 [3].

2.2.6 *Color-based detection*

Tracking suatu objek menggunakan warna merupakan salah satu metode yang paling cepat dan mudah. Pembatasan warna dapat dilakukan dengan memilih *range threshold* yang sesuai. Cara ini dilakukan dengan mengecek setiap piksel dari gambar input dan hanya yang bernilai diatas batas nilai *threshold* yang dimunculkan. Dalam pembatasannya jika menggunakan HSV maka ditetapkan suatu nilai *hue* yang paling rendah dan paling tinggi untuk di deteksi (H_{high} dan H_{low}), kemudian ditetapkan juga nilai *value* dan *saturation*nya (V_{high}, V_{low}, S_{high}, dan S_{low}) sehingga hasil akhirnya akan terbentuk adalah *binary image* (citra biner) dimana *foreground* berwarna putih dan *background* berwarna hitam. Citra biner berarti setiap piksel akan disimpan dalam bentuk single bit (0 atau 1) dan ini berpengaruh pada kecepatan *image processing*. Fungsi karakteristik dari *foreground* yang terdeteksi dalam citra biner di definisikan sebagai berikut :

$$I_M(I,j) \begin{cases} = 1 & for\ foreground \\ = 0 & for\ background \end{cases} \quad \text{.....(8)}$$

Dimana $I_M(I,j)$ adalah citra biner dengan posisi pixel (I,j) [12]

2.2.7 Open CV

Open CV adalah sebuah *open source* untuk *library computer vision* yang bisa didapatkan dari http://SourceForge.net/projects/opencvlibrary. Library ini ditulis menggunakan Bahasa C dan C++ dan dapat dioperasikan menggunakan linux, Windows, dan Mac OS X. sudah dikembangkan juga tampilan menggunakan Phyton, Ruby, Matlab,

dan bahasa lainnya. OpernCV didesain untuk efisiensi komputasi dan berfokus pada aplikasi *real time*. Salah satu tujuan dari OpenCV adalah untuk menyediakan infrastruktur *computer vision* yang mudah digunakan sehingga dapat membantu orang-orang dalam membuat aplikasi vision dengan cepat. Library OpenCV berisi lebih dari 500 fungsi yang mencakup banyak area dalam *vision*, termasuk di dalamnya *medical imaging*, *security*, *user interface*, kalibrasi kamera, *stereo vision*, dan robotika [13].

2.2.8 *Exponential Smoothing*

Exponential smoothing termasuk dalam salah satu cara *forcasting*. Dimana pada setiap data baru yang dimasukkan akan menggantikan observasi yang lalu dan menghitung prediksi yang baru. Alasan dinamakan *exponential smoothing* adalah karena tiap *increment* yang lalu di kurangi dengan (1-α). Metode *exponential smoothing* hanya memerlukan tiga jenis data untuk memprediksi data selanjutnya, yaitu prediksi lalu, data aktual lalu dan konstanta *smoothing* alfa (α). Konstanta *smoothing* ini menentukan level dari *smoothing* juga kecepatan reaksi antara prediksi dan data aktual yang muncul. Persamaan untuk *single exponential smoothing* adalah sebagai berikut [14].

$$F_t = F_{t-1} + \alpha(A_{t-1} - F_{t-1}) \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (9)$$

Dimana :

F_t = prediksi untuk periode t
F_{t-1} = prediksi untuk periode sebelum t
A_{t-1} = data aktual pada periode sebelum t
α = konstanta *smoothing* atau laju respon yang diinginkan

2.2.9 Filter Kalman

Di tahun 1960, Kalman mempublikasikan makalahnya yang menjelaskan sebuah persoalan penyaringan linier data diskrit [Kalman, 60]. Sejak saat itu, Kalman Filter menjadi topik

penelitian dan terapan yang luas, terutama di bidang navigasi [15].

2.2.9.1 Proses yang diestimasi

Persoalan umum untuk Kalman Filter diskrit adalah mencoba untuk mengestimasi state dari sebuah proses waktu diskrit yang dinyatakan oleh persamaan beda stokastik linier.

$$x_k = Ax_{k-1} + Bu_{k-1} + w_{k-1} \quad \text{.....(10)}$$

dengan pengukuran yang dinyatakan :

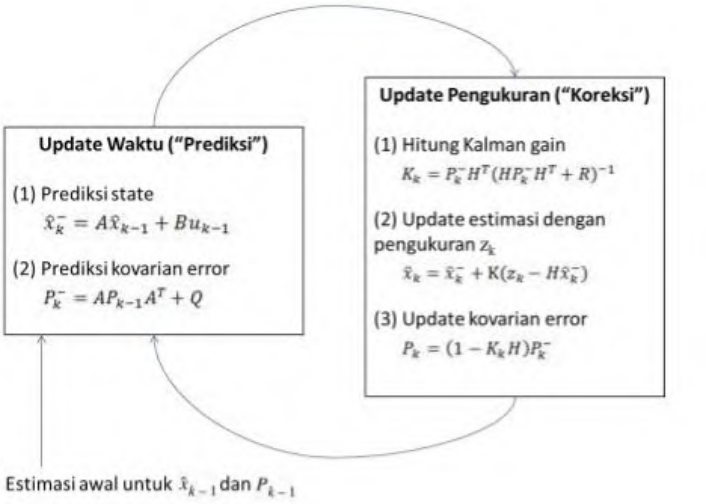
$$z_k = Hx_k + v_k \quad \text{.....(11)}$$

wk dan vk adalah variabel acak yang mewakili *noise* proses dan *noise* pengukuran, keduanya independen, jenis *white noise* (nilainya tetap), dengan probabilitas berdistribusi normal. Matriks A (n×n) dalam persamaan (10) menghubungkan *state* pada waktu diskrit sebelumnya, yaitu k-1, dengan *state* pada waktu diskrit sekarang, yaitu k, tanpa pengaruh fungsi pemicu u atau *noise* proses w. Dalam praktik, A bisa berubah dalam tiap waktu, tapi di sini kita asumsikan konstan. Matriks B (n×l) menghubungkan input kontrol u dengan *state* x, u bersifat opsional (bisa ada / tidak). Matriks H (m×n) dalam persamaan pengukuran (11) menghubungkan *state* dengan pengukuran z_k.

2.2.9.2 Proses Kalman Filter

Tugas pertama dalam *Update* pengukuran adalah menghitung *Kalman Gain* (Kk). Selanjutnya mengukur nilai proses aktual z_k, kemudian menghitung pasca-estimasi state dengan melibatkan nilai hasil pengukuran. Terakhir adalah mendapatkan nilai pasca-estimasi kovarian error. Setelah menjalani satu siklus update waktu dan pengukuran, siklus ini diulang yang mana nilai pasca-estimasi sebelumnya digunakan

untuk memprediksi nilai pra-estimasi yang baru. Sifat rekursif ini adalah satu sifat penting dari Kalman filter seperti terlihat pada gambar 2.6.



Gambar 2.6 Skema lengkap operasi kalman filter [15].

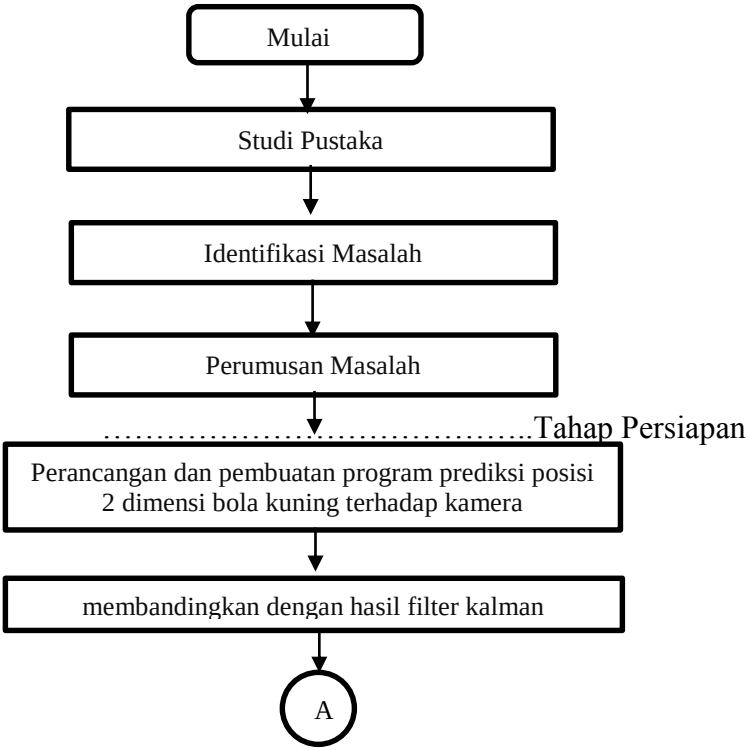
(Halaman ini sengaja dikosongkan)

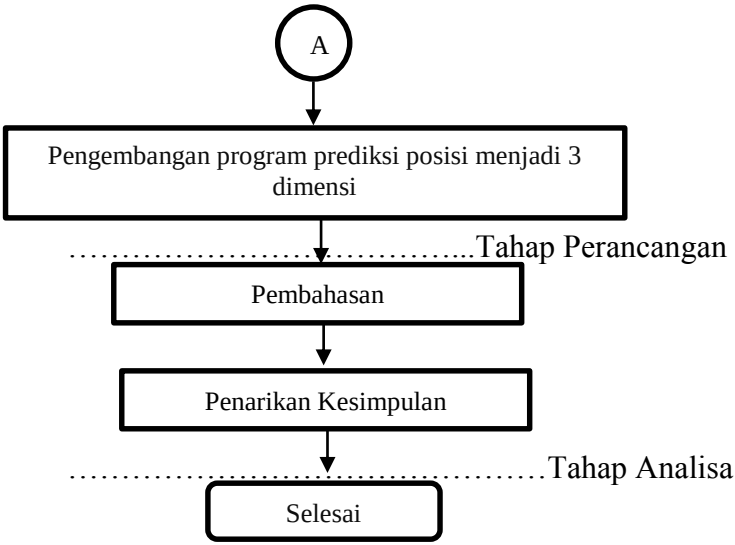
BAB III
METODE PENELITIAN

Pada Bab III ini akan dipaparkan mengenai langkah-langkah sistematis yang akan dijadikan sebagai acuan kerangka penelitian. Selain itu juga akan dijelaskan parameter proses yang digunakan dalam penelitian ini.

3.1 Diagram Alir Metodologi Perancangan

Secara garis besar metodologi yang digunakan dalam penelitian ini dapat dibagi menjadi tiga tahapan utama yaitu tahap persiapan, tahap perancangan program, serta tahap analisa yang berisi pembahasan dan penarikan kesimpulan.





Gambar 3.1 *Flowchart* metodologi penelitian

3.2 Tahap Persiapan

Tahap persiapan ini terdiri dari tiga tahapan yaitu studi pustaka, identifikasi masalah dan perumusan masalah. Studi pustaka adalah acuan referensi bagi penulis untuk memahami permasalahan yang akan diteliti mengenai pembuatan aplikasi program prediksi, pendeteksi, pelacak dan penentu koordinat 3 dimensi suatu objek dari buku-buku referensi dan jurnal yang berkaitan dengan permasalahan yang akan dibahas. Kemudian dilakukan identifikasi masalah untuk menyusun sistem prediksi, pendeteksi, pelacak, dan penentu koordinat 3 dimensi suatu objek. Sedangkan perumusan masalah mencangkup perancangan program untuk prediksi, pendeteksi, pelacak, dan penentu koordinat 3 dimensi suatu objek.

3.3 Tahap Perancangan Program

Pada tahap ini dilakukan perancangan program prediksi posisi 2 dimensi terlebih dahulu menggunakan metode *exponential*

smoothing, lalu di bandingkan hasilnya mengunakan metode filter kalman, selanjutnya program prediksi dikembangkan menjadi prediksi 3 dimensi,dan di terapkan pada sistem senjata.

3.3.1 Cara Kerja

Langkah kerja pertama yang harus dilakukan adalah mendeteksi objek yang merupakan bola berwarna kuning dalam bentuk image secara *real time*. Kemudian citra ini dianalisa dan diproses menggunakan program C++ dari *library* openCV 2.4.10 dan *Software* Visual Studio 2015. Dari hasil tangkapan ini kemudian di dapat koordinat 2 Dimensi dari bola kuning, kemudian dengan menggunakan persamaan *exponential smoothing* didapat prediksi posisi bola, kita menghitung *error root mean square*nya (prediksi - hasil sebenarnya) hasil ini akan dibandingkan dengan menggunakan *error root mean square* filter kalman, jadi prediksi menggunakan filter kalman juga dilakukan. Ketika perbedaan kedua *error* tadi tidak lebih dari 0,1 maka, akan dikembangkan program prediksi 3 dimensi dari bola kuning menggunakan *exponential smoothing*.

3.3.2 Proses dan Pengerjaan

Ada beberapa tahapan dalam proses dan pengerjaan tugas pada perancangan program yaitu pendektesian dan pelacakan objek. Kemudian menentukan posisi 2 dimensi bola, lalu menerapkan persamaan *exponential smoothing* agar dapat memprediksi koordinat x, y selanjutnya. Membandingkan hasil *error* dengan hasil *error* yang didapat menggunakan metode filter kalman, baru setelah itu menggunakan persamaan *exponential smoothing* untuk prediksi 3 dimensi.

1. Pendektesian, Pelacakan dan Penetuan koordinat x, y objek

   Pendektesian dan Pelacakan Objek dilakukan dengan cara mengatur nilai warna sesuai yang diinginkan pada program agar pada saat program dijalankan bisa mendeteksi dan melacak objek yang memiliki warna tertentu. Langkah – langkah yang perlu dilakukan yaitu mengambil citra yang

didalamnya terdapat objek tertentu. Kemudian dilakukan proses untuk mengubah citra yang sebelumnya RGB ke HSV. Citra yang sudah berupa HSV dilakukan *thresholding* untuk menghasilkan citra biner dimana warna kuning menjadi warna hitam dan warna lainnya menjadi warna putih. Kemudian di dapatkan titik tengah dari objek, titik tengah ini mewakili posisi benda, sehingga hasil yang kita dapat adalah posisi x dan y dalam piksel.

2. Penentuan prediksi dan membandingkan hasil koordinat x, y objek

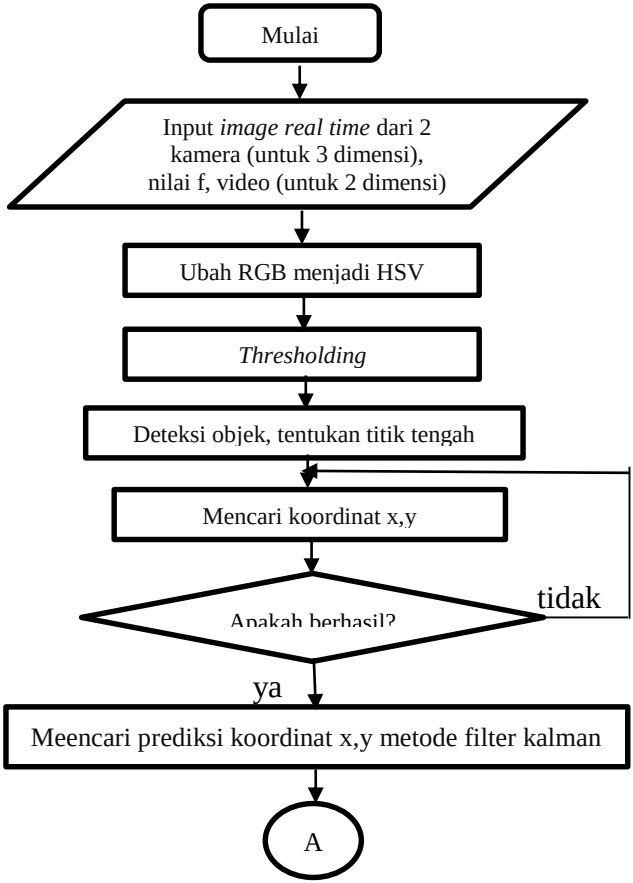
Setelah mendapatkan posisi x dan y, kita mengaplikasikan rumus *exponential smoothing* untuk mendapatkan prediksi posisi x dan y, lalu menghitung *Error root mean square*nya. Hasil ini nantinya akan dibandingkan dengan menggunakan hasil *error* dari prediksi metode filter kalman. Ketika kedua hasil menunjukkan perbedaan 0,1 atau kurang maka baru kita mengaplikasikan *exponential smoothing* untuk prediksi 3 dimensi objek. *Input* untuk proses pembandingan ini adalah berupa video, agar kedua proses, baik menggunakan *exponensial smoothing* maupun filter kalman mempunyai *input*an yang sama.

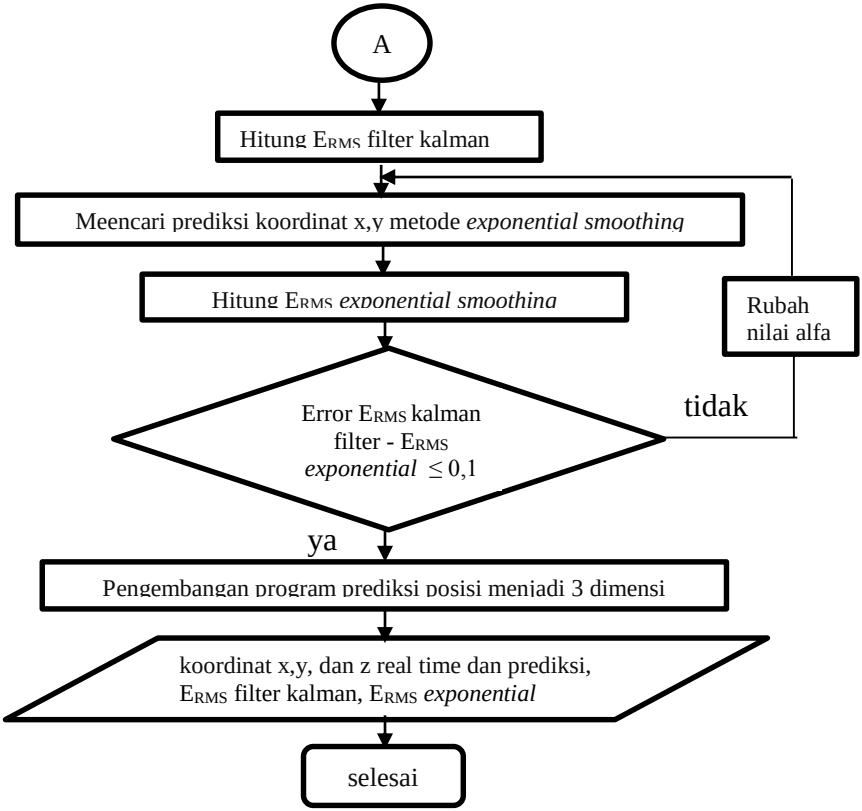
3. Penentuan prediksi posisi x, y, dan z objek

Penentuan koordinat x, y dan z objek terhadap 2 kamera dilakukan dengan memanfaatkan nilai hasil dari kalibrasi (nilai *focal length* kamera). Adapun cara kalibrasi adalah dengan memasukkan gambar papan catur yang jarak antar kotaknya sudah diketahui kemudian di lakukan proses *remap* untuk mensejajarkan hasil tangkapan gambar dari 2 buah kamera. Kalibrasi harus dilakukan pada awal pengambilan data. Sehingga jika objek digerakkan menjauh maupun mendekat terhadap kamera jarak akan secara otomatis diketahui. Jika koordinat z sudah diketahui, koordinat x dan y bisa ditentukan juga dengan menggunakan persamaan. Setelah mendapatkan

koordinat x, y, dan z dilakukan prediksi koordinat selanjutnya menggunakan metode *exponential smoothing* dengan nilai alfa yang ditetapkan sebelumnya dari hasil pembandingan dengan filter kalman. Nantinya koordinat x, y dan z dari objek terhadap 2 kamera diserialkan agar bisa diproses untuk menjadi data masukan pada Alat Pelontar Peluru

3.3.3 *Flowchart* Tahap Perancangan Program





Gambar 3.2 *Flowchart* perancangan program

3.4 Tahap Implementasi

Tahap Implementasi merupakan tahap implementasi dari *software* yang telah dirancang. *Running* program dengan berbagai variasi posisi objek. Kemudian dari hasil deteksi objek tersebut sehingga didapatkan data berupa koordinat x, y dan z objek baik secara real time dan juga prediksi dari berbagai posisi. Data ini dijadikan bahan analisa untuk menentukan persamaan hubungan antara koordinat objek terhadap 2 kamera dan koordinat objek terhadap laras Alat Pelontar Peluru.

BAB IV
KONSTRUKSI PROGRAM

4.1 Implementasi Program

Program pendeteksi objek ini dibuat dengan menggunakan teknologi *Computer Vision*. Program ini menggunakan perangkat lunak sebagai berikut :

a. OpenCV versi 2.4.10
b. Microsoft Visual Studio 2015

Pengambilan beberapa data berupa video diperlukan untuk membandingkan hasil *error* dari program penentuan prediksi koordinat menggunakan metode *exponential smoothing* dengan metode filter kalman. Pengujian ini dilakukan pada kondisi tertentu yaitu :

a. Objek yang diambil berupa bola berdiameter ± 5 cm.
b. Pengambilan objek di dalam ruangan dan pencahayaan *indoor*

Sedangkan mesin pengolah yang digunakan dalam pengujian ini yaitu komputer dengan spesifikasi sebagai berikut:

a. *Processor Intel Core i5*
b. *Memory RAM* 4,00 GB
c. *System Type 64-Bit Operating system*
d. *Operating system Windows* 8.1

4.2 Konstruksi Program

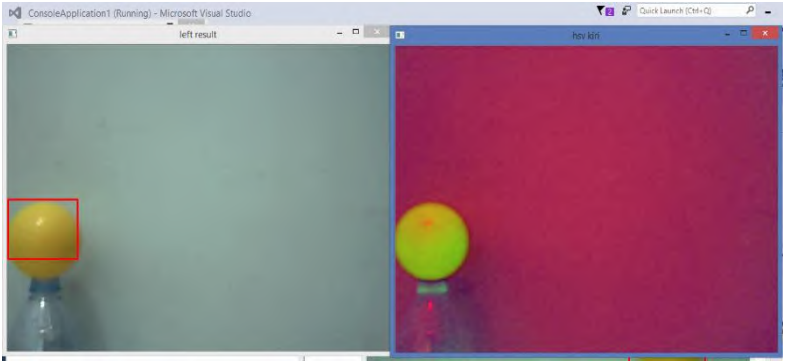
Pada tahap kontruksi ini dibahas tentang tahap-tahap penerapan metode yang digunakan dalam pembuatan program penetuan prediksi koordinat 3 dimensi untuk objek tunggal. Proses kontruksi ini dimulai dari tahap pendeteksi objek, kemudian dilakukan penetuan koordinat x, y, z objek, lalu penentuan prediksi koordinat x, y, z. Kontruksi program dibangun ke dalam kode program menggunakan *tools* yang telah ditentukan. Program yang digunakan untuk menerapkan metode tersebut kedalam kode program adalah Microsoft Visual Studio 2015 dengan *library OpenCV* dengan bahasa pemrograman *C++*.

4.2.1 Program Pendeteksian Objek

Kode program 1: *RGB to HSV*

```
IplImage *HSV=cvCreateImage(cvSize(frame->width,frame->height),8,3);
IplImage *HSV2=cvCreateImage(cvSize(frame2->width,frame2->height),8,3);
cvCvtColor(frame,HSV,CV_BGR2HSV);
cvCvtColor(frame2,HSV2,CV_BGR2HSV);
```

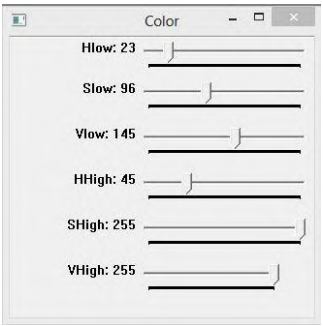
Pada proses pendeteksian objek proses yang dilakukan pertama yaitu mengubah citra masukan RGB, yaitu gambar yang pertama ditangkap oleh kamera berformat RGB menjadi citra HSV yang dilakukan dengan menggunakan kode program. Hal ini dilakukan karena hasil *thresholding* yang diperoleh pada citra HSV akan lebih baik dibanding dengan hasil yang diperoleh dengan melakukan proses *thresholding* pada citra RGB. *Frame* hasil yang ditangkap kamera USB berbentuk dalam format citra RGB (*Red Gren Blue*) dengan resolusi citra 640 x 480 *pixel*. Hasil dari *frame* tersebut akan diproses menjadi citra HSV dapat dilihat pada gambar 4.1.



Gambar 4.1 Gambar RGB menjadi HSV kamera kiri

Kode program 2: Mencari range warna *threshold*

```
cvNamedWindow("Color",1);
cvCreateTrackbar("Hlow","Color", &hl,179,0);
cvCreateTrackbar("Slow","Color", &sl,255,0);
cvCreateTrackbar("Vlow","Color", &vl,255,0);
cvCreateTrackbar("HHigh","Color", &hh,179,0);
cvCreateTrackbar("SHigh","Color", &sh,255,0);
cvCreateTrackbar("VHigh","Color", &vh,255,0);
```



Gambar 4.2 *Trackbar* untuk menentukan *range thresholding*

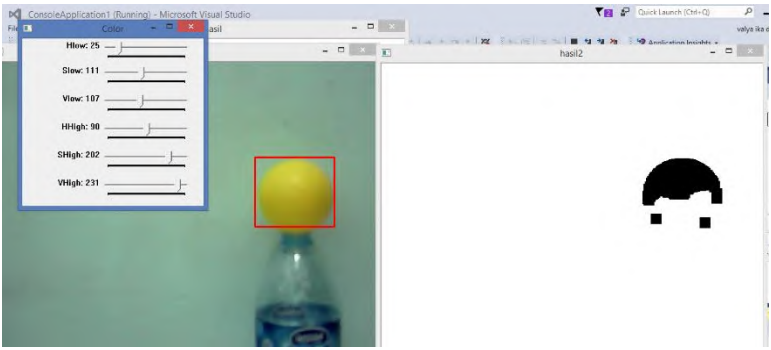
Dari citra HSV yang telah diperoleh, kemudian dilakukan proses *thresholding*, dengan cara menentukan *hue* maksimal, *hue* minimal, *saturation* maksimal, *saturation* minimal, *value* maksimal, dan *value* minimal. Untuk mencari rentang tersebut digunakan *trackbar* seperti gambar 4. Sehingga perubahan *trackbar* dan perubahan hasil *threshold* dapat dilihat pada gambar 4. Dari proses diatas didapat rentang *threshold* sebagai berikut: *Hue* minimum = 23, *Hue* maksimum = 45, *Saturation* minimum = 96, *Saturation* maksimum = 255, *Value* minimum = 180, *Value* maksimum = 255.

Hasil dari *range* yang diperoleh kemudian digunakan untuk proses pendeteksian objek yaitu *thresholding*. Proses pendeteksian sangat bergantung pada hasil pengolahan citra yang sudah dilakukan, sehingga penentuan range ini harus dilakukan dengan beberapa percobaan untuk mendapatkan hasil

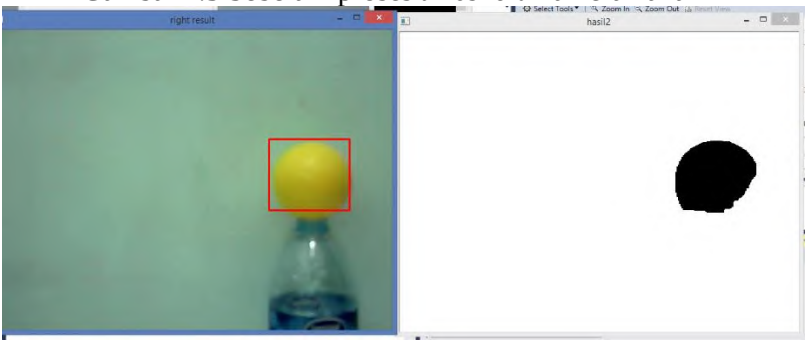
yang terbaik. Kondisi lingkungan saat dilakukan percobaan juga sangat berpengaruh, sehingga nilai rentang yang diperoleh sewaktu – waktu bisa berubah sesuai kondisi pencahayaan ruangan tempat pengujian dilakukan.

Kode program 3: *Threshold*

```
IplImage *thres=cvCreateImage(cvSize(frame->width,frame-
>height),8,1);
IplImage *thres2=cvCreateImage(cvSize(frame2->width,frame2-
>height),8,1);
cvInRangeS(HSV, cvScalar(hl,sl , vl), cvScalar(hh, sh, vh), thres);
cvInRangeS(HSV2, cvScalar(hl,sl , vl), cvScalar(hh, sh, vh), thres2);
```



Gambar 4.3 Sebelum proses *threshold* kamera kanan



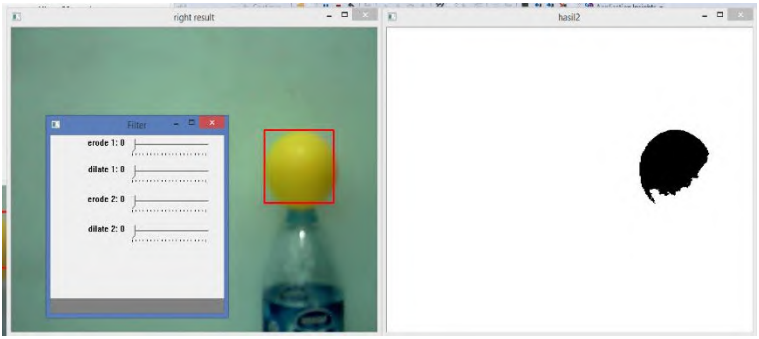
Gambar 4.4 Setelah proses *threshold* kamera kanan

Kemudian dilakukan proses *threshold* pada citra hasil proses yang berupa citra HSV. Tipe *threshold* yang digunakan adalah *binary threshold* dengan rentang nilai HSV yang telah didapatkan sebelumnya, yaitu HSV minimum (23, 96, 180) dan HSV maksimum (45, 255 255). Proses dapat dilihat pada gambar 4.2 dan gambar 4.3

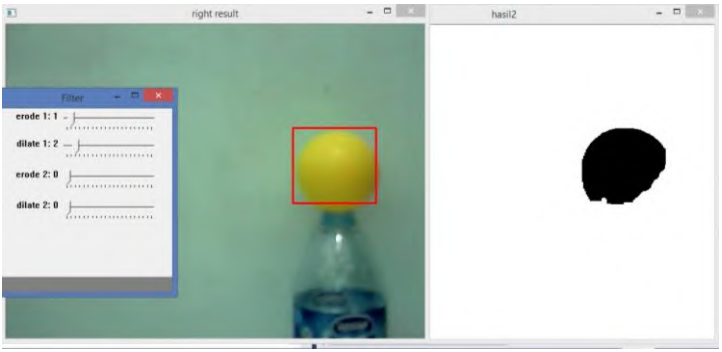
Kode program 4: *Erode dan dilate*

```
cvErode(thres,thres,element,ero1);
cvDilate(thres,thres,element,dil1);
cvErode(thres,thres,element,ero2);
cvDilate(thres,thres,element,dil2);
```

Pada proses *threshold* ini diperlukan proses *noise filtering* karena pada proses *threshold* biasanya terdapat gangguan/*noise* pada hasil citra yang diperoleh. *Noise filtering* bertujuan untuk membuang atau mengurangi gangguan tersebut agar citra threshold yang diperoleh sesuai dengan yang diinginkan. *Noise filtering* yang diakai terdiri dari proses *erode* dan *dilate*, dimana *erode* bertujuan untuk mengurangi *pixel* sedangkan *dilate* bertujuan untuk menambah *pixel* (gambar 4.4 dan gambar 4.5).



Gambar 4.5 Sebelum proses *erode* dan *dilate*



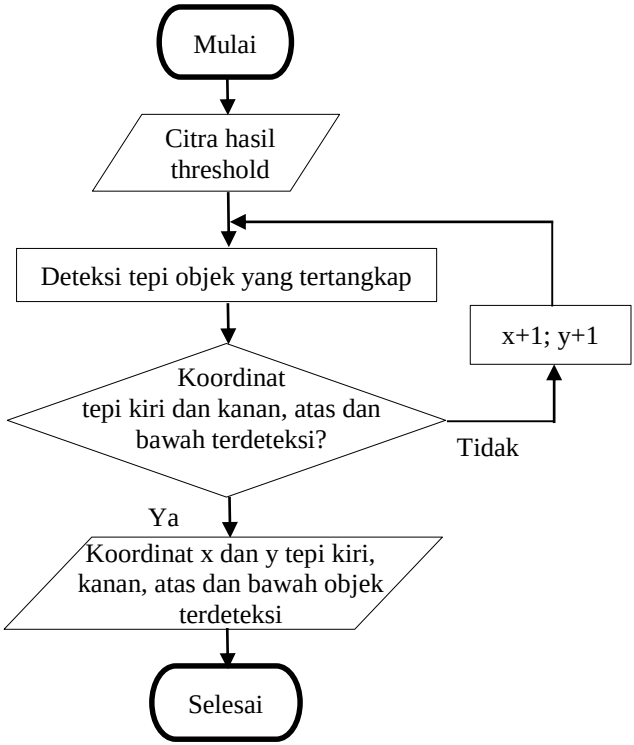
Gambar 4.6 Setelah proses *erode* dan *dilate*

Setelah semua proses tersebut dilakukan dilanjutkan dengan proses pendeteksian. Pendeteksian objek dilakukan dengan menggunakan cara pendeteksian warna. Setelah dilakukan proses *threshold* untuk mendapatkan citra biner, dimana objek yang dideteksi akan berwarna hitam dan selain objek akan berwarna putih atau menjadi *background*, maka koordinat x dan y dari titik tengah objek dapat ditemukan.

Kode program 5: Mendeteksi objek

```
for(int x=0;x<thres->width;x=x++)
for (int y=0;y<thres->height;y=y++)
        {if(hasil->imageData[hasil->widthStep*y+x*hasil-
>nChannels]==0)
                {ukur.x= x;
                }}

for (int y=0;y<thres->height;y=y++)
for(int x=0;x<thres->width;x=x++)
        {if(hasil->imageData[hasil->widthStep*y+x*hasil-
>nChannels]==0)
                {ukur.y= y;
                }}
```



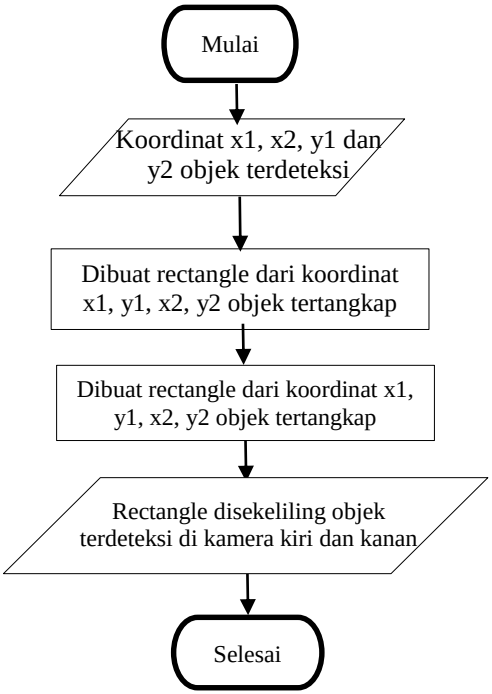
Gambar 4.7 Diagram alir pencarian koordinat tepi objek

Dalam pendeteksian objek tertentu melalui warna dapat dilakukan dengan memanfaatkan hasil dari proses *threshold* yang sudah dilakukan terlebih dahulu. Yaitu program akan mendeteksi dari kiri atas menuju ke kanan lalu berpindah ke baris piksel berikutnya pada *binary frame* apakah terjadi perubahan nilai piksel, pada saat bersamaan program mendeteksi dari kanan bawah menuju ke kiri lalu berpindah ke baris piksel diatasnya, jika terjadi perubahan nilai piksel berarti itu menunjukkan koordinat dari tepi objek. Dari program tersebut nantinya akan diperoleh koordinat x dan y tepi kiri dan

kanan objek, sehingga titik tengah dari objek tersebut bisa diketahui melalui perhitungan.

Kode program 6: Menunjukkan objek yang terdeteksi

```
cvRectangle( frame,cvPoint( ukur.x, ukur.y),cvPoint(  ukur1.x , ukur1.y),cvScalar( 0, 0, 255, 0 ), 2, 0, 0 );
cvRectangle( frame2,cvPoint( ukur2.x, ukur2.y),cvPoint(  ukur3.x , ukur3.y),cvScalar( 0, 0, 255, 0 ), 2, 0, 0 );
```



Gambar 4.8 Diagram alir pembuatan *rectangle*

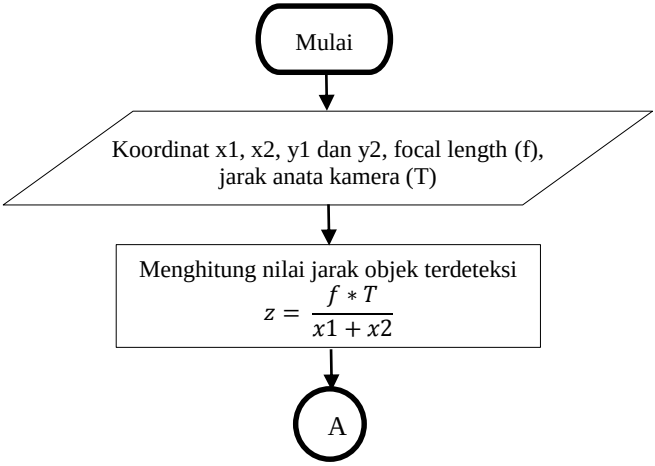
Setelah diperoleh titik koordinat tepi kiri dan kanan dari objek, dibuat program yang bertujuan untuk membuat persegi disekeliling objek. Hal itu bertujuan untuk menunjukkan lokasi

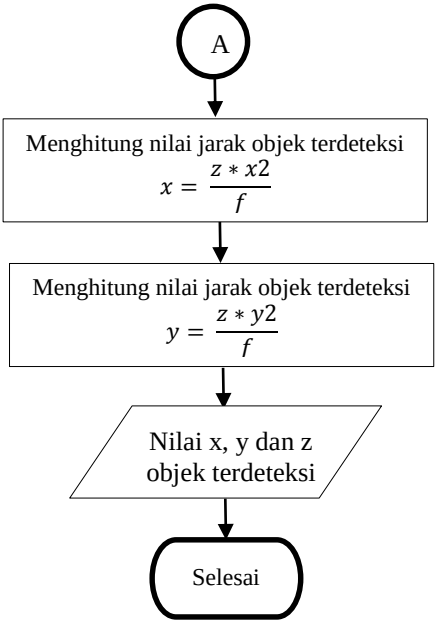
dimana objek yang sudah terdeteksi. Sehingga pada saat program dijalankan maka secara otomatis benda berwarna tertentu sesuai yang diinginkan terdeteksi dan secara otomatis akan ditandai dengan kotak merah disekeliling objek yang terdeteksi tersebut.

4.2.2 Program Penentuan Koordinat x, y, dan z

Kode program 7 : Menetukan koordinat z, x, dan y

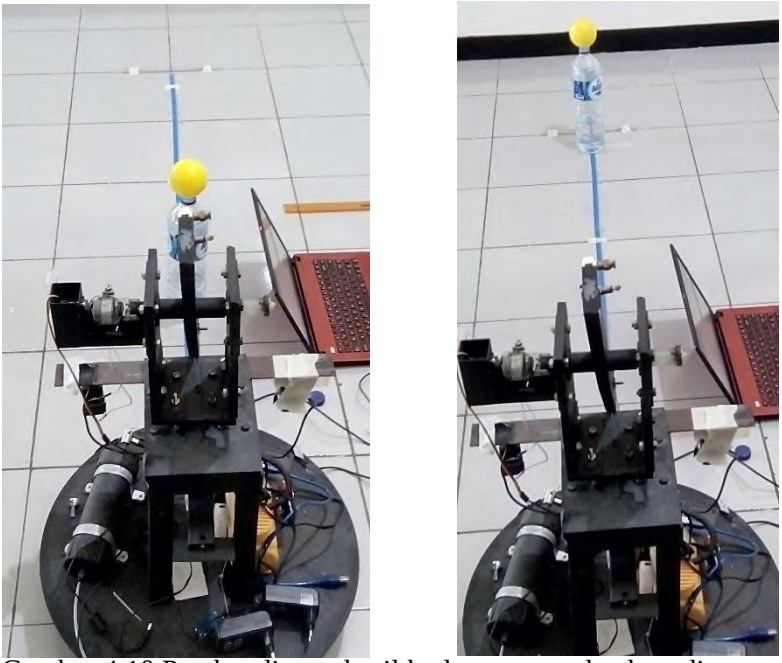
```
int xfix=((ukur.x-ukur1.x)/2)+ukur1.x;
int yfix=((ukur.y-ukur1.y)/2)+ukur1.y;
int xfix2=(((ukur2.x-ukur3.x)/2)+ukur3.x);
int yfix2=(((ukur2.y-ukur3.y)/2)+ukur3.y);
int f=8.1876385326098966e+002;
int d=((640-xfix2)+xfix);
int s=abs(300*f)/d;
int z=s+((0.0088*(s*s))-(4.3533*s)+392.55);
int t=(xfix2*z)/f;
int x=(t-((0.473*t)+71.077))+15;
int u=(yfix2*z)/f;
int y=(u+((-1.4594*u)+31.092));
```





Gambar 4.9 Diagram alir penentuan nilai x, y, z objek terhadap kamera

Untuk mencari nilai koordinat z atau jarak objek terhadap kamera dipakai koordinat titik tengah dari objek yang terdeteksi di kamera kanan dan kamera kiri. Dari kedua koordinat titik tengah tersebut dibuat persamaan untuk menemukan nilai z. Persamaan yang digunakan yaitu persamaan (1). Sedangkan untuk mencari koordinat x dan y yang merupakan jarak vertikal dan horizontal objek terhadap kamera dapat digunakan persamaan (2) dan (3) pada bab 2.



Gambar 4.10 Pembandingan hasil kode program dan koordinat sebenarnya

Setelah dicoba untuk mengetahui hasil dari program, masih didapat hasil yang tidak sesuai, misalkan jarak yang harusnya 180cm masih terbaca 220cm, begitu juga pada koordinat x dan y. Hal ini bisa disebabkan kualitas kamera, pencahayaan, dan lain sebagainya. Untuk menanggulanginya dilakukan percobaan dan membandingkan antara hasil koordinat pada program dan koordinat sebenarnya, lalu dibuat suatu persamaan error menggunakan microsoft excel, sehingga ketika persamaan ini dimasukkan hasil koordinat pada program dan koordinat sebenarnya tidak berbeda jauh.

4.2.3. Penentuan Prediksi Koordinat

Kode program 8 : Rekam dan *load* video

```
CvSize size = cvSize((int)cvGetCaptureProperty( capture,
CV_CAP_PROP_FRAME_WIDTH),
                    (int)cvGetCaptureProperty( capture,
CV_CAP_PROP_FRAME_HEIGHT));
CvVideoWriter* writer1=
cvCreateVideoWriter("kiri.avi",CV_FOURCC('M','J','P','G'), 5,size);

CvVideoWriter*writer2=cvCreateVideoWriter("kanan.avi",CV_FOURCC
('M','J','P','G'), 5,size);
CvCapture* capture, *capture2;
          if (pilihanVid) {
                    capture = cvCreateFileCapture("kiri.avi");
                    capture2 = cvCreateFileCapture("kanan.avi");}
          else {
                    capture = cvCaptureFromCAM(1);
                    capture2 = cvCaptureFromCAM(2);}
```

Untuk membandingkan hasil antara prediksi menggunakan filter kalman dan dengan menggunakan *exponential smoothing* harus dengan inputan yang sama sehingga inputan yang dipilih berupa video. Beberapa video diambil untuk merepresentasikan kejadian nyata atau secara *real time*.

Kode program 9 : Penentuan prediksi koordinat x, y *kalman filter*

```
CvKalman* kalman = cvCreateKalman(4, 2, 0);
const CvMat* state = cvCreateMat(4, 1, CV_32FC1);
CvMat* measurement = cvCreateMat(2, 1, CV_32FC1);
cvZero(measurement);
CvMat* processnoise = cvCreateMat(4, 1, CV_32FC1);
cvSetIdentity(kalman->transition_matrix);
cvSetIdentity(kalman->measurement_matrix);
kalman->measurement_matrix->data.fl[0] = 1;
kalman->measurement_matrix->data.fl[5] = 1;
kalman->process_noise_cov->data.fl[0] = 0.0005;
kalman->process_noise_cov->data.fl[5] = 0.0005;
kalman->process_noise_cov->data.fl[10] = 0.0005;
kalman->process_noise_cov->data.fl[15] = 0.0005;
cvSetIdentity(kalman->measurement_noise_cov, cvScalar(0.1));
double ticks;
bool klik_ya = 0, found_ya = 1;
int notFound;
measurement->data.fl[0] = x_x;
measurement->data.fl[1] = y_y;
if (!found_ya) {
kalman->error_cov_pre->data.fl[0] = 1;
kalman->error_cov_pre->data.fl[5] = 1;
kalman->error_cov_pre->data.fl[10] = 1;
kalman->error_cov_pre->data.fl[15] = 1;
state->data.fl[0] = measurement->data.fl[0];
state->data.fl[1] = measurement->data.fl[1];
state->data.fl[2] = 0;
state->data.fl[3] = 0;
found_ya = 1;
```

Filter kalman digunakan sebagai pembanding penentuan prediksi menggunakan *exponential smoothing*, didalam kalman filter parameter yang harus dimasukkan adalah nilai prediksi awal dan matrik covariance. Parameter yang digunakan mengacu pada jurnal berjudul "*Kalman Tracking for Image Processing"* oleh Julius Oyeleke.

Kode program 11 : Penentuan prediksi koordinat x, y, dan z *exponential smoothing*

```
float alpha = 0.4995;
                predX = predX + alpha*(xs - predX);prevX = predX;
                predY = predY + alpha*(ys - predY);prevY = predY;
                predZ = predZ + alpha*(z - predZ);prevZ = predZ;
```

Penentuan prediksi koordinat x, y, dan z dengan metode *exponential smoothing* didapat dengan rumus (9) pada bab 2. Metode ini hanya memerlukan nilai alfa dan nilai prediksi sebelumnya dalam prosesnya, untuk itu harus dicari terlebih dahulu nilai alfa yang sesuai dengan cara membandingkan hasil metode filter kalman dan *exponential smoothing*.

BAB V
ANALISA DAN PEMBAHSAN

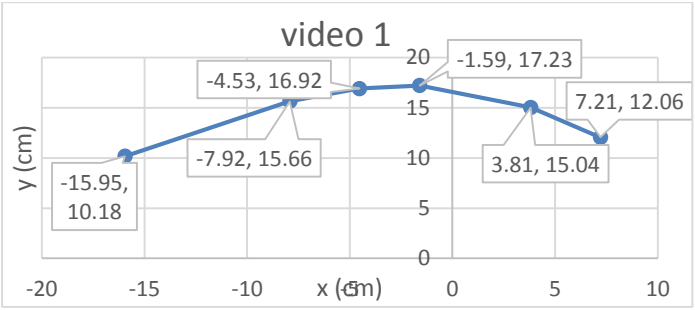
5.1 Pengambilan Data

Untuk mencari nilai alfa yang sesuai langkah pertama yang dilakukan adalah mengambil data. Pengambilan data dilakukan dengan cara merekam video pergerakan sebuah bola berwarna kuning yang dilemparkan pada suatu bidang miring hingga bola bergerak membentuk sebuah kurva, dengan jarak dari kamera adalah ± 160cm. Data diambil dengan selang waktu 0,2 detik. Untuk merepresentasikan keadaan sebenarnya (*real time*), maka ada 10 video yang diambil dengan ketentuan jumlah data yang terekam lebih dari 5 data. Berikut adalah data x, y, z *real time* yang didapat dari 10 video seperti pada tabel 5.1 untuk video 1, untuk video 2 hingga 10 dapat dilihat pada lampiran A. Selain itu dapat juga dilihat perbedaan lintasan pada 10 video yang ada pada gambar 5.2 sampai gambar 5.11

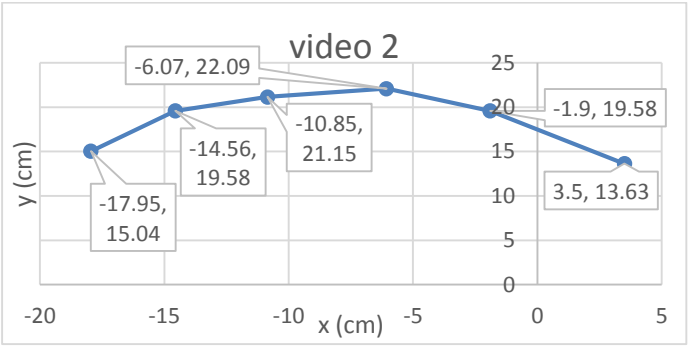
Gambar 5.1 *Setting* pengambilan data

Tabel 5.1 Data koordinat z, x, y real time video 1.

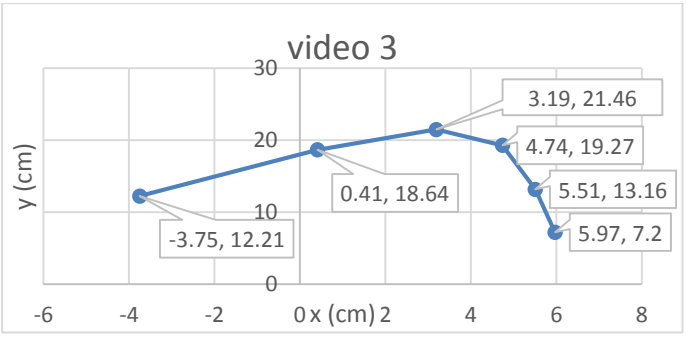
z	x	y
165.33	7.21	12.06
166.76	3.81	15.04
176.51	-1.59	17.23
167.48	-4.53	16.92
158.48	-7.92	15.66
164.63	-15.95	10.18



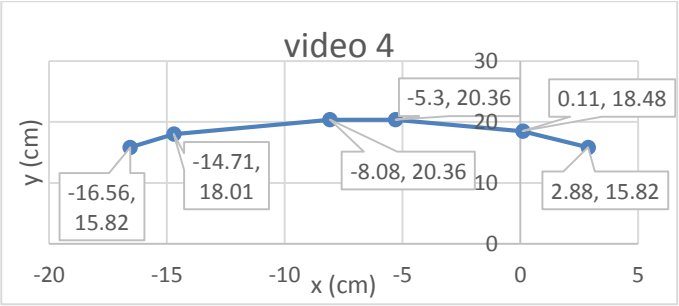
Gambar 5.2 Lintasan bola pada video 1



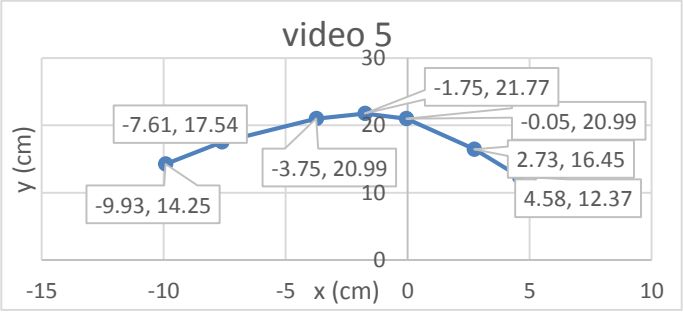
Gambar 5.3 Lintasan bola pada video 2



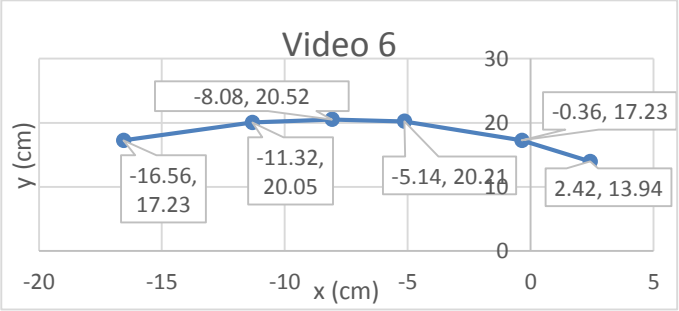
Gambar 5.4 Lintasan bola pada video 3



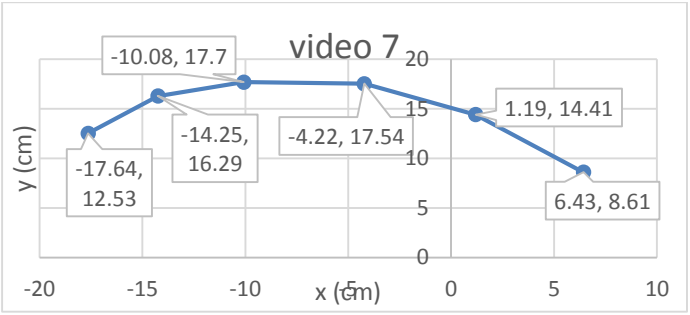
Gambar 5.5 Lintasan bola pada video 4



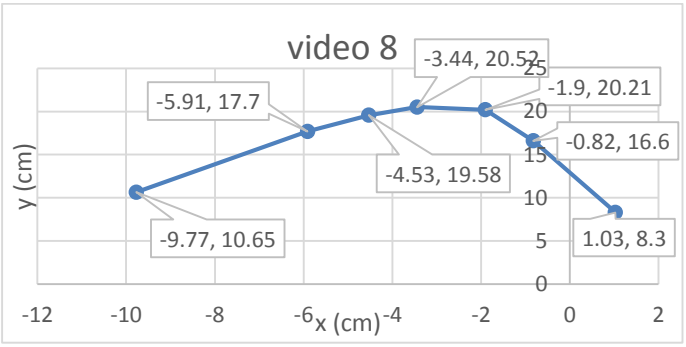
Gambar 5.6 Lintasan bola pada video 5



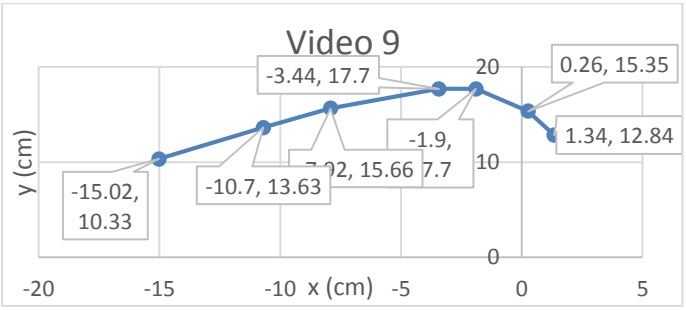
Gambar 5.7 Lintasan bola pada video 6



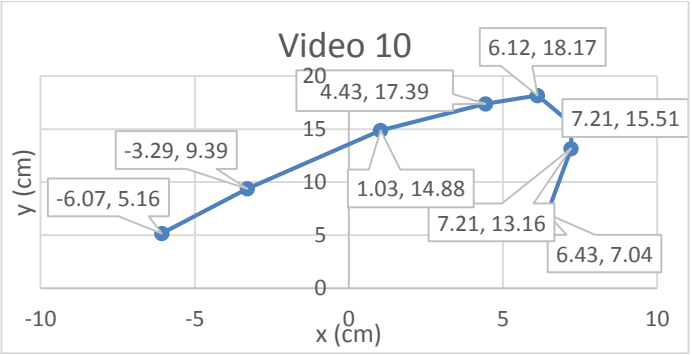
Gambar 5.8 Lintasan bola pada video 7



Gambar 5.9 Lintasan bola pada video 8



Gambar 5.10 Lintasan bola pada video 9



Gambar 5.11 Lintasan bola pada video 11

5.2 Pencarian Nilai Alfa

Nilai alfa didapat dari membandingkan nilai *error root mean square* dari metode filter kalman dan juga *error root mean square* dari metode *exponential smoothing*. *Error* sendiri adalah selisih antara nilai prediksi dan nilai sebenarnya yang terjadi. Jika selisih ERMS dari koordinat x dan y *exponential smoothing* lebih kecil atau sama 0.1 dengan nilai ERMS koordinat x dan y filter kalman, maka nilai alfa dianggap benar perhitungan nilai alfa dapat dilihat pada tabel 5.2

Tabel 5.2 Perhitungan alfa pada video 1.

		kalman filter				exponential smoothing				
x	y	predict x	predict y	error x	error y	alfa	predict x	predict y	error x	error y
7.21	12.06	0	0	0	0	0.405				
3.81	15.04	0.66	1.1	-3.15	-13.94	0.405	2.9200	4.8843	-0.89	-10.156
-1.59	17.23	6.96	28.97	8.55	11.74	0.405	3.2804	8.99735	4.8704	-8.2326
-4.53	16.92	-6.99	19.42	-2.46	2.5	0.405	1.3079	12.3315	5.8379	-4.5884
-7.92	15.66	-7.46	16.6	0.46	0.94	0.405	-1.056	14.1898	6.8635	-1.4701
-15.9	10.18	-11.32	14.41	4.63	4.23	0.405	-3.836	14.7852	12.113	4.60528
0	0	-23.97	4.69	0	0	0.405	-8.742	12.9201	0	0
ERMS				4.70585	8.45195				7.10561	6.56259
ERMS x dan y				6.840342					6.839492	

Setelah menghitung error yaitu selisih antara prediksi dan nilai sebenarnya dengan kedua metode, didapatkan error masing-masing data baik koordinat x maupun koordinat y. Lalu untuk menyatakan error yang berbeda-beda nilainya pada tiap data menjadi 1 nilai, dipilih metode ERMS yaitu dengan mengkuadratkan hasil error tiap data yang didapat kemudian menjumlahkannya dan dibagi dengan jumlah data yng diperoleh setelah itu diakar. Untuk metode kalman filter didapatkan ERMS sebesar 4.70585cm untuk koordinat x dan 8.45195cm untuk koordinat y, kemudian untuk metode exponential smoothing didapatkan ERMS untuk koordinat x sebesar 7.10561 dan koordinat y sebesar 6.56259.

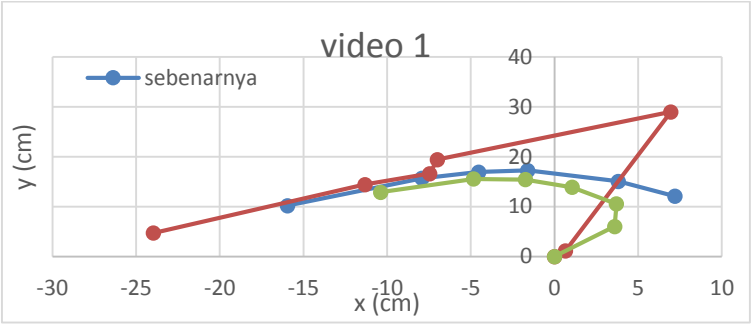
Karena masih terdapat 2 nilai yang berbeda yaitu ERMS koordinat x dan ERMS koordinat y, maka dilakukan satu kali lagi proses ERMS untuk mendapatkan 1 nilai error dari keduanya yaitu ERMS x dan y. Seperti pada tabel 5.11 ERMS x dan y metode kalman filter sebesar 9.6737 dan ERMS x dan y metode exponential smoothing sebesar 9.6725, karena selilisih ERMS kedua metode ini kurang dari 0,1 maka nilai alfa untuk video 1 adalah sebesar 0.405. Dengan cara yang sama didapatkan nilai alfa dari video 2 hingga 10 bertutut-turut sebesar : 0.415; 0.577; 0.357; 0.436; 0.425; 0.6285; 0.435; 0.79; 0.527. Sehingga nilai rata-rata alfa adalah 4.995, nilai ini yang akan digunakan untuk mencari prediksi koordinat x, y, dan z metode *exponential smoothing*.

5.3 Pembahasan dan Analisa

Dari nilai alfa yang diperoleh yaitu sebesar 0.49955, maka gambar grafik dibawah ini menunjukkan posisi bola sebenarnya, prediksi kalman filter, dan prediksi menggunakan metode *exponential smoothing* (gambar 5.2 Sampai 5.11). Serta data yang diperoleh tampak pada tabel 5.3 hingga tabel 5.12.

Tabel 5.3 Koordinat x dan y *real time, exponential smoothing* dan filter kalman pada video 1.

sebenarnya		kalman filter				exponential smoothing			
x	y	predict x	predict y	error x	error y	predict x	predict y	error x	error y
7.21	12.06	0	0	0	0	0	0		
3.81	15.04	0.66	1.1	-3.15	-13.9	3.6018	6.0246	-0.208	-9.015
-1.59	17.23	6.96	28.97	8.55	11.74	3.7058	10.528	5.2958	-6.702
-4.53	16.92	-6.99	19.42	-2.46	2.5	1.0603	13.876	5.5903	-3.044
-7.92	15.66	-7.46	16.6	0.46	0.94	-1.732	15.397	6.1877	-0.263
-16	10.18	-11.3	14.41	4.63	4.23	-4.823	15.528	11.127	5.3482
		-24	4.69	0	0	-10.38	12.857	0	0
ERMS				4.71	8.452			6.6547	5.7294
ERMS x dan y				6.8403				6.209333	

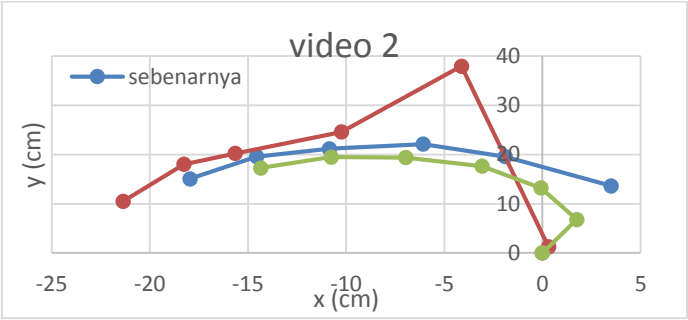


Gambar 5.12 : Posisi bola sebenarnya, prediksi kalman dan *exponential smoothing* pada video 1

Pada tabel 5.3 dapat dilihat bahwa error terbesar metode filter kalman koordinat x sebesar 8.55cm terjadi pada saat prediksi kedua, dan error terbesar untuk koordinat y adalah sebesar 13.9cm terjadi saat prediksi pertama. Untuk metode *exponential smoothing* error terbesar untuk koordinat x adalah 11.127cm pada prediksi kelima dan untuk koordinat y sebesar 9.015cm pada saat prediksi pertama.

Tabel 5.4 Koordinat x dan y *real time*, *exponential smoothing* dan filter kalman pada video 2.

sebenarnya		kalman filter				exponential smoothing			
x	y	predict x	predict y	error x	error y	predict x	predict y	error x	error y
3.5	13.63	0	0	0	0	0	0		
-1.9	19.58	0.32	1.24	2.22	-18.3	1.7484	6.8089	3.6484	-12.77
-6.07	22.09	-4.12	37.92	1.95	15.83	-0.074	13.189	5.9959	-8.901
-10.9	21.15	-10.2	24.59	0.61	3.44	-3.069	17.635	7.7806	-3.515
-14.6	19.58	-15.6	20.21	-1.08	0.63	-6.956	19.391	7.6038	-0.189
-18	15.04	-18.3	18.01	-0.31	2.97	-10.75	19.485	7.1953	4.4455
		-21.4	10.49	0	0	-14.35	17.265	0	0
ERMS				1.44	11.03			6.624	7.4093
ERMS x dan y				7.8636				7.027631	

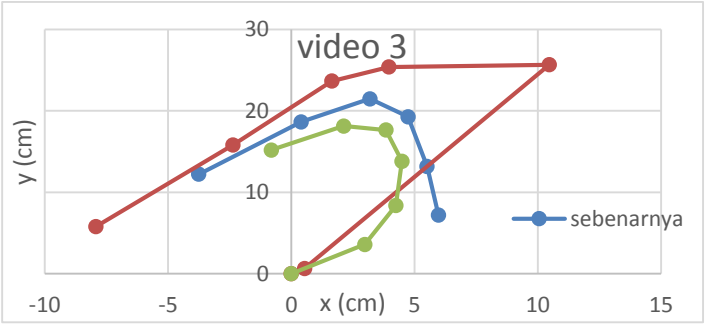


Gambar 5.13 : Posisi bola sebenarnya, prediksi kalman dan *exponential smoothing* pada video 2

Pada tabel 5.4 dapat dilihat bahwa error terbesar metode filter kalman koordinat x sebesar 2.22cm terjadi pada saat prediksi kedua, dan error terbesar untuk koordinat y adalah sebesar 18.3cm terjadi saat prediksi pertama. Untuk metode *exponential smoothing* error terbesar untuk koordinat x adalah 7.78cm pada prediksi ketiga dan untuk koordinat y sebesar 12.77cm pada saat prediksi pertama.

Tabel 5.5 Koordinat x dan y *real time*, *exponential smoothing* dan filter kalman pada video 3.

sebenarnya		kalman filter				exponential smoothing			
x	y	predict x	predict y	eror x	eror y	predict x	predict y	eror x	eror y
5.97	7.2	0	0	0	0	0	0		
5.51	13.16	0.54	0.65	-4.97	-12.5	2.9823	3.5968	-2.528	-9.563
4.74	19.27	10.5	25.66	5.73	6.39	4.245	8.3741	-0.495	-10.9
3.19	21.46	3.96	25.38	0.77	3.92	4.4923	13.817	1.3023	-7.643
0.41	18.64	1.65	23.65	1.24	5.01	3.8417	17.635	3.4317	-1.005
-3.75	12.21	-2.36	15.82	1.39	3.61	2.1274	18.137	5.8774	5.9271
		-7.92	5.79	0	0	-0.809	15.176	0	0
ERMS				3.51	7.083			3.3061	7.8068
ERMS x dan y				5.5895				5.994843	

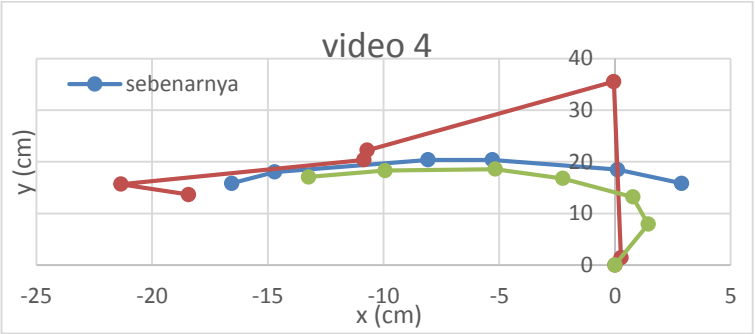


Gambar 5.14 : Posisi bola sebenarnya, prediksi kalman dan *exponential smoothing* pada video 3

Pada tabel 5.5 dapat dilihat bahwa error terbesar metode filter kalman koordinat x sebesar 5.73cm terjadi pada saat prediksi kedua, dan error terbesar untuk koordinat y adalah sebesar 12.5cm terjadi saat prediksi pertama. Untuk metode *exponential smoothing* error terbesar untuk koordinat x adalah 5.8874cm pada prediksi kelima dan untuk koordinat y sebesar 9.563cm pada saat prediksi pertama.

Tabel 5.6 Koordinat x dan y *real time*, *exponential smoothing* dan filter kalman pada video 4.

sebenarnya			kalman filter			exponential smoothing			
x	y	predict x	predict y	error x	error y	predict x	predict y	error x	error y
2.88	15.82	0	0	0	0	0	0		
0.11	18.48	0.26	1.44	0.15	-17	1.4387	7.9029	1.3287	-10.58
-5.3	20.36	-0.05	35.53	5.25	15.17	0.7749	13.187	6.0749	-7.173
-8.08	20.36	-10.7	22.24	-2.62	1.88	-2.26	16.77	5.8202	-3.59
-14.7	18.01	-10.9	20.36	3.86	2.35	-5.167	18.563	9.5427	0.5534
-16.6	15.82	-21.4	15.66	-4.79	-0.16	-9.934	18.287	6.6257	2.467
		-18.4	13.63	0	0	-13.24	17.055	0	0
ERMS				3.802	10.29			6.4422	6.0434
ERMS x dan y				7.758		6.246			

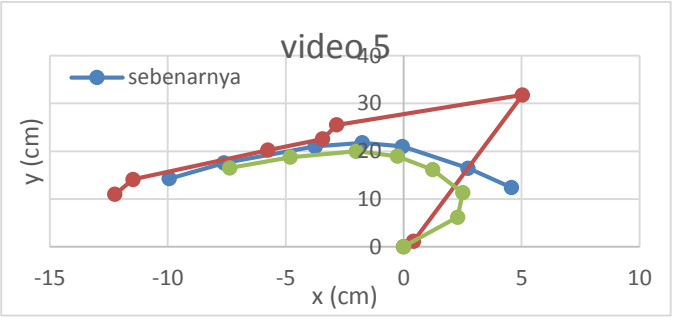


Gambar 5.15 : Posisi bola sebenarnya, prediksi kalman dan *exponential smoothing* pada video 4

Pada tabel 5.6 dapat dilihat bahwa error terbesar metode filter kalman koordinat x sebesar 5.25cm terjadi pada saat prediksi kedua, dan error terbesar untuk koordinat y adalah sebesar 17cm terjadi saat prediksi pertama. Untuk metode *exponential smoothing* error terbesar untuk koordinat x adalah 9.5427cm pada prediksi keempat dan untuk koordinat y sebesar 10.58cm pada saat prediksi pertama.

Tabel 5.7 Koordinat x dan y *real time, exponential smoothing* dan filter kalman pada video 5.

sebenarnya		kalman filter				exponential smoothing			
x	y	predict x	predict y	error x	error y	predict x	predict y	error x	error y
4.58	12.37	0	0	0	0	0	0		
2.73	16.45	0.42	1.12	-2.31	-15.3	2.2879	6.1794	-0.442	-10.27
-0.05	20.99	5.04	31.77	5.09	10.78	2.5088	11.31	2.5588	-9.68
-1.75	21.77	-2.83	25.53	-1.08	3.76	1.2305	16.146	2.9805	-5.624
-3.75	20.99	-3.44	22.56	0.31	1.57	-0.258	18.955	3.4916	-2.035
-7.61	17.54	-5.76	20.21	1.85	2.67	-2.003	19.972	5.6074	2.4317
-9.93	14.25	-11.5	14.1	-1.54	-0.15	-4.804	18.757	5.1262	4.507
		-12.2	10.96	0	0	-7.365	16.506	0	0
ERMS				2.527	7.905			3.7758	6.5978
ERMS x dan y				5.8685				5.37527	

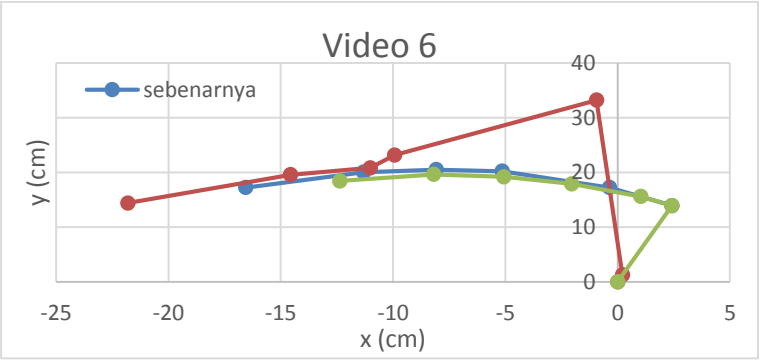


Gambar 5.16 : Posisi bola sebenarnya, prediksi kalman dan *exponential smoothing* pada video 5

Pada tabel 5.7 dapat dilihat bahwa error terbesar metode filter kalman koordinat x sebesar 5.09cm terjadi pada saat prediksi kedua, dan error terbesar untuk koordinat y adalah sebesar 15.3cm terjadi saat prediksi pertama. Untuk metode *exponential smoothing* error terbesar untuk koordinat x adalah 5.6074cm pada prediksi kelima dan untuk koordinat y sebesar 10.27cm pada saat prediksi pertama.

Tabel 5.8 Koordinat x dan y *real time*, *exponential smoothing* dan filter kalman pada video 6.

sebenarnya		kalman filter				exponential smoothing			
x	y	predict x	predict y	error x	error y	predict x	predict y	error x	error y
2.42	13.94	0	0	0	0	0	0		
-0.36	17.23	0.22	1.27	0.58	-16	2.42	13.94	2.78	-3.29
-5.14	20.21	-0.94	33.19	4.2	12.98	1.0313	15.584	6.1713	-4.626
-8.08	20.52	-9.93	23.18	-1.85	2.66	-2.052	17.895	6.0284	-2.625
-11.3	20.05	-11	20.83	0.31	0.78	-5.063	19.206	6.2569	-0.844
-16.6	17.23	-14.6	19.58	2	2.35	-8.189	19.628	8.3713	2.3977
		-21.8	14.41	0	0	-12.37	18.43	0	0
ERMS				2.258	9.342			6.1868	3.0193
ERMS x dan y				6.7963				6.078	

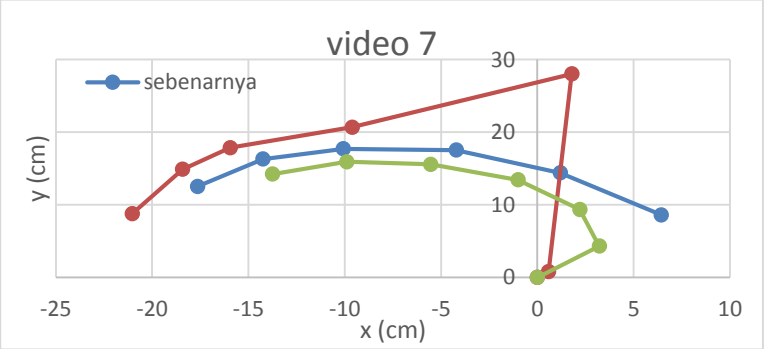


Gambar 5.17 : Posisi bola sebenarnya, prediksi kalman dan *exponential smoothing* pada video 6

Pada tabel 5.8 dapat dilihat bahwa error terbesar metode filter kalman koordinat x sebesar 4.2cm terjadi pada saat prediksi kedua, dan error terbesar untuk koordinat y adalah sebesar 16cm terjadi saat prediksi pertama. Untuk metode *exponential smoothing* error terbesar untuk koordinat x adalah 8.3713cm pada prediksi kelima dan untuk koordinat y sebesar 4.626cm pada saat prediksi kedua.

Tabel 5.9 Koordinat x dan y *real time*, *exponential smoothing* dan filter kalman pada video 7.

sebenarnya		kalman filter				exponential smoothing			
x	y	predict x	predict y	error x	error y	predict x	predict y	error x	error y
6.43	8.61	0	0	0	0	0	0		
1.19	14.41	0.58	0.78	-0.61	-13.6	3.2121	4.3011	2.0221	-10.11
-4.22	17.54	1.79	28.03	6.01	10.49	2.202	9.351	6.422	-8.189
-10.1	17.7	-9.62	20.68	0.46	2.98	-1.006	13.442	9.0739	-4.258
-14.3	16.29	-16	17.86	-1.7	1.57	-5.539	15.569	8.711	-0.721
-17.6	12.53	-18.4	14.88	-0.78	2.35	-9.891	15.929	7.7494	3.3992
		-21	8.77	0	0	-13.76	14.231	0	0
ERMS					2.836	7.908		7.2609	6.3159
ERMS x dan y					5.9404			6.8048	

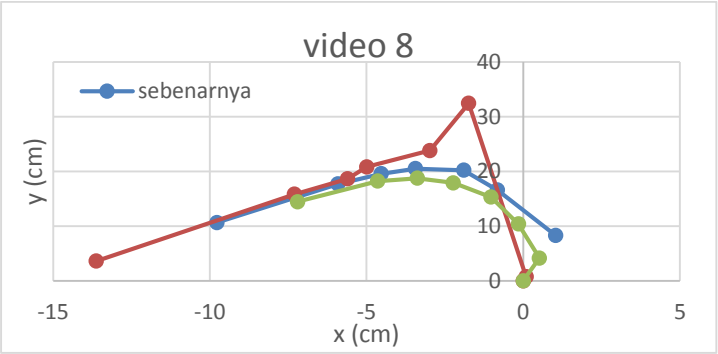


Gambar 5.18 : Posisi bola sebenarnya, prediksi kalman dan *exponential smoothing* pada video 7

Pada tabel 5.9 dapat dilihat bahwa error terbesar metode filter kalman koordinat x sebesar 6.01cm terjadi pada saat prediksi kedua, dan error terbesar untuk koordinat y adalah sebesar 13.6cm terjadi saat prediksi pertama. Untuk metode *exponential smoothing* error terbesar untuk koordinat x adalah 9.0739cm pada prediksi ketiga dan untuk koordinat y sebesar 10.11cm pada saat prediksi pertama.

Tabel 5.10 Koordinat x dan y *real time, exponential smoothing* dan filter kalman pada video 8.

sebenarnya		kalman filter				Exponential smoothing			
x	y	predict x	predict y	error x	error y	predict x	predict y	error x	error y
1.03	8.3	0	0	0	0	0	0		
-0.82	16.6	0.09	0.75	0.91	-15.9	0.5145	4.1463	1.3345	-12.45
-1.9	20.21	-1.74	32.45	0.16	12.24	-0.152	10.368	1.7479	-9.842
-3.44	20.52	-2.98	23.81	0.46	3.29	-1.025	15.284	2.4147	-5.236
-4.53	19.58	-4.99	20.83	-0.46	1.25	-2.232	17.9	2.2984	-1.68
-5.91	17.7	-5.61	18.64	0.3	0.94	-3.38	18.739	2.5303	1.0391
-9.77	10.65	-7.3	15.82	2.47	5.17	-4.644	18.22	5.1263	7.57
		-13.6	3.6	0	0	-7.205	14.438	0	0
ERMS				1.116	8.574			2.8469	7.5343
ERMS x dan y				6.1136				5.695	

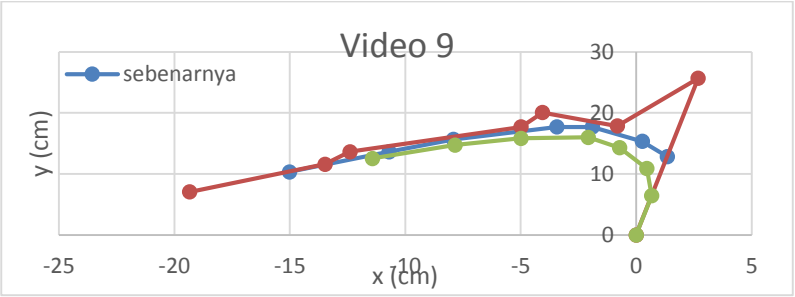


Gambar 5.19 : Posisi bola sebenarnya, prediksi kalman dan *exponential smoothing* pada video 8

Pada tabel 5.10 dapat dilihat bahwa error terbesar metode filter kalman koordinat x sebesar 2.47cm terjadi pada saat prediksi keenam, dan error terbesar untuk koordinat y adalah sebesar 15.9cm terjadi saat prediksi pertama. Untuk metode *exponential smoothing* error terbesar untuk koordinat x adalah 5.1263cm pada prediksi keenam dan untuk koordinat y sebesar 12.45cm pada saat prediksi pertama.

Tabel 5.11 Koordinat x dan y *real time*, *exponential smoothing* dan filter kalman pada video 9.

sebenarnya		kalman filter				exponential smoothing			
x	y	predict x	predict y	error x	error y	predict x	predict y	error x	error y
1.34	12.84	0	0	0	0	0	0		
0.26	15.35	2.68	25.68	2.42	10.33	0.6694	6.4142	0.4094	-8.936
-1.9	17.7	-0.82	17.86	1.08	0.16	0.4649	10.878	2.3649	-6.822
-3.44	17.7	-4.06	20.05	-0.62	2.35	-0.716	14.286	2.7235	-3.414
-7.92	15.66	-4.99	17.7	2.93	2.04	-2.077	15.991	5.843	0.3315
-10.7	13.63	-12.4	13.63	-1.7	0	-4.996	15.826	5.7041	2.1959
-15	10.33	-13.5	11.59	1.54	1.26	-7.845	14.729	7.1746	4.3989
		-19.3	7.04	0	0	-11.43	12.531	0	0
ERMS					1.882	4.435		4.6785	5.2014
ERMS x dan y					3.4066			4.9468	

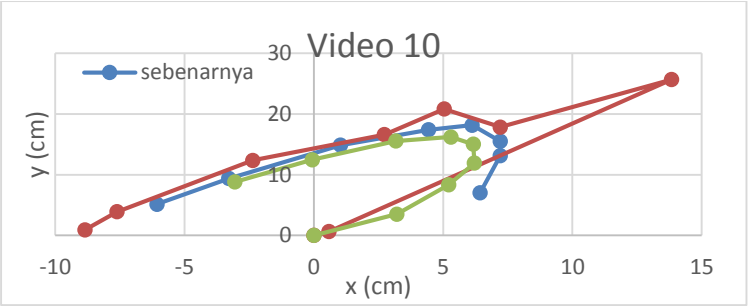


Gambar 5.20 : Posisi bola sebenarnya, prediksi kalman dan *exponential smoothing* pada video 9

Pada tabel 5.11 dapat dilihat bahwa error terbesar metode filter kalman koordinat x sebesar 2.42cm terjadi pada saat prediksi pertama, dan error terbesar untuk koordinat y adalah sebesar 10.33cm terjadi saat prediksi pertama. Untuk metode *exponential smoothing* error terbesar untuk koordinat x adalah 7.1746cm pada prediksi keenam dan untuk koordinat y sebesar 8.936cm pada saat prediksi pertama.

Tabel 5.12 Koordinat x dan y *real time*, *exponential smoothing* dan filter kalman pada video 10.

sebenarnya		kalman filter				exponential smoothing			
x	y	predict x	predict y	error x	error y	predict x	predict y	error x	error y
6.43	7.04	0	0	0	0	0	0		
7.21	13.16	0.58	0.64	-6.63	-12.5	3.2121	3.5168	-3.998	-9.643
7.21	15.51	13.8	25.67	6.62	10.16	5.2093	8.3341	-2.001	-7.176
6.12	18.17	7.21	17.86	1.09	-0.31	6.2087	11.919	0.0887	-6.251
4.43	17.39	5.04	20.83	0.61	3.44	6.1644	15.042	1.7344	-2.348
1.03	14.88	2.73	16.6	1.7	1.72	5.298	16.215	4.268	1.3347
-3.29	9.39	-2.36	12.37	0.93	2.98	3.1659	15.548	6.4559	6.158
-6.07	5.16	-7.61	3.91	-1.54	-1.25	-0.059	12.472	6.0109	7.3118
		-8.85	0.93	0	0	-3.062	8.8192	0	0
ERMS				3.693	6.384			4.1236	6.3498
				5.2152				5.354	



Gambar 5.21 : Posisi bola sebenarnya, prediksi kalman dan *exponential smoothing* pada video 10

Pada tabel 5.12 dapat dilihat bahwa error terbesar metode filter kalman koordinat x sebesar 6.63cm terjadi pada saat prediksi pertama, dan error terbesar untuk koordinat y adalah sebesar 12.5cm terjadi saat prediksi pertama. Untuk metode *exponential smoothing* error terbesar untuk koordinat x adalah 6.4559cm pada prediksi keenam dan untuk koordinat y sebesar 9.643cm pada saat prediksi pertama.

Dari gambar 5.1 hingga 5.10 didapatkan lintasan yang berbeda tiap gambarnya, dikarenakan bola dilempar dengan menggunakan tangan, sehingga pasti berbeda kekuatan melempar bola dari video 1 hingga 10, kemudian diambil sebanyak 10 kali untuk merepresentasikan keadaan nyata. Dengan nilai alfa 0.49955 didapatkan nilai selisih ERMS untuk koordinat x dan y antara metode *exponensial smoothing* dengan metode filter kalman berturut-turut dari video 1 hingga video 10 sebesar : -0.63101, -0.83596, 0.405324, -1.51205, -0.49323, -0.71769, 0.864395, -0.41838, 1.540231, 0.138499. Dimana jika hasil selisih adalah negatif berarti metode *exponential smoothing* memiliki nilai Erms yang lebih sedikit daripada metode filter kalman. Pada gambar 5.1 Sampai 5.10 diatas dapat dilihat perbedaan yang sangat besar antara prediksi metode filter kalman dan dengan metode *exponential smoothing*.

Hal ini dikarenakan pada metode kalman filter memperhitungkan *noise* pengukuran dan *noise* sistem, menghitung matrik kovarian, dan juga kalman gain, semua proses ini dilakukan secara rekursif dan dapat berbeda nilainya. Misalkan untuk kalman gain sendiri dapat berubah nilainya dari prediksi data 1 dengan prediksi data 2 sehingga filter kalman dapat menyesuaikan prediksinya. Untuk metode *exponential smoothing* sendiri hanya bergantung pada data, prediksi sebelumnya, dan nilai alfa yang diinputkan. Sehingga metode kalman filter membutuhkan setidaknya 6 data untuk dapat mengikuti pergerakan bola yang sebenarnya, sedangkan metode *exponential smoothing* sudah dapat mengikuti pergerakan bola yang sebenarnya dari awal, namun untuk alfa 0,4995 hasil yang diinginkan masih terlambat atau kurang mendahului pergerakan bola.

Selisih terbesar antara prediksi dan nilai sebenarnya banyak terjadi pada prediksi data ke dua atau tiga. Hal ini disebabkan tidak adanya prediksi awal, diasumsikan prediksi awal

adalah 0,0 padahal dari awal data pergerakan bola dimulai dari ujung kiri frame yang bernilai positif paling besar sehingga untuk mengejar pergerakan bola membutuhkan waktu yang cukup lama.

5.4 Data Koordinat 3 Dimensi Sebenarnya dan Prediksi *Exponential Smoothing*

Tabel 5.13 Data Koordinat 3 Dimensi Sebenarnya dan Prediksi Exponential Smoothing video 1

sebenarnya			exponential smoothing					
z	x	y	predict x	predict y	predict z	eror x	eror y	error z
165.33	7.21	12.06	0	0	160			
166.76	3.81	15.04	3.60176	6.02457	162.663	-0.208	-9.015	-4.0974
176.51	-1.59	17.23	3.70578	10.5282	164.709	5.2958	-6.702	-11.8005
167.48	-4.53	16.92	1.06028	13.8761	170.604	5.5903	-3.044	3.124418
158.48	-7.92	15.66	-1.7323	15.3967	169.044	6.1877	-0.263	10.56362
164.63	-16	10.18	-4.8234	15.5282	163.767	11.127	5.3482	-0.86344
			-10.382	12.8565	164.198	0	0	
Erms						6.6547	5.7294	7.458404
Erms x, y, z						6.6518		

Dengan didapatkan nilai alfa maka prediksi 3 dimensi bisa diketahui. Seperti tabel 5.14 diatas dan juga dapat dihitung Erms x, y, z dari video 1-10 yang datanya dapat dilihat pada lampiran D. dari data yang telah diperoleh diketahui bahwa Erms x, y, z dari prediksi exponential smoothing dari video 1 hingga 10 bernilai :6.6518; 7.0759; 5.7342; 7.1494; 6.1449; 6.9632; 6.1738; 5.8343; 4.7383; 6.3647.

Lampiran

Lampiran A. Data Koordinat 3 Dimensi Real Time

Tabel A.1 Data koordinat z, x, y sebenarnya video 2

z	x	y
172.66	3.5	13.63
166.76	-1.9	19.58
155.87	-6.07	22.09
161.17	-10.85	21.15
155.87	-14.56	19.58
147.82	-17.95	15.04
0	0	0

Tabel A.2 Data koordinat z, x, y sebenarnya video 3

z	x	y
170.41	5.97	7.2
165.33	5.51	13.16
163.23	4.74	19.27
163.93	3.19	21.46
164.63	0.41	18.64
175.73	-3.75	12.21
0	0	0

Tabel A.3 Data koordinat z, x, y sebenarnya video 4

z	x	y
173.42	2.88	15.82
157.82	0.11	18.48
171.15	-5.3	20.36
160.49	-8.08	20.36
167.48	-14.71	18.01
152.69	-16.56	15.82
0	0	0

Tabel A.4 Data koordinat z, x, y sebenarnya video 5

z	x	y
173.42	4.58	12.37
174.95	2.73	16.45
165.33	-0.05	20.99
157.82	-1.75	21.77
155.23	-3.75	20.99
167.48	-7.61	17.54
166.76	-9.93	14.25
0	0	0

Tabel A.5 Data koordinat z, x, y sebenarnya video 6

z	x	y
176.51	2.42	13.94
159.82	-0.36	17.23
169.67	-5.14	20.21
161.17	-8.08	20.52
153.95	-11.32	20.05
164.63	-16.56	17.23
0	0	0

Tabel A.6 Data koordinat z, x, y sebenarnya video 7

z	x	y
164.63	6.43	8.61
165.33	1.19	14.41
163.23	-4.22	17.54
165.33	-10.08	17.7
172.66	-14.25	16.29
163.23	-17.64	12.53
0	0	0

Tabel A.7 Data koordinat z, x, y sebenarnya video 8

z	x	y
187.17	1.03	8.3
165.33	-0.82	16.6
161.17	-1.9	20.21
160.49	-3.44	20.52
159.82	-4.53	19.58
159.82	-5.91	17.7
169.67	-9.77	10.65
0	0	0

Tabel A.8 Data koordinat z, x, y sebenarnya video 9

z	x	y
174.18	1.34	12.84
159.15	0.26	15.35
163.23	-1.9	17.7
158.48	-3.44	17.7
164.63	-7.92	15.66
159.15	-10.7	13.63
167.48	-15.02	10.33
0	0	0

Tabel A.9 Data koordinat z, x, y sebenarnya video 10

z	x	y
168.93	6.43	7.04
161.85	7.21	13.16
159.15	7.21	15.51
160.49	6.12	18.17
161.85	4.43	17.39
161.85	1.03	14.88
176.51	-3.29	9.39
178.88	-6.07	5.16
0	0	0

B. Data perhitungan nilai alfa dan prediksi x, y exponential smoothing

Tabel B.1 Perhitungan alfa pada video 2

		kalman filter				exponential smoothing				
x	y	predict x	predict y	error x	error y	alfa	predict x	predict y	error x	error y
3.5	13.63	0	0	0	0	0.415				
-1.9	19.58	0.32	1.24	2.22	-18.34	0.415	1.4525	5.65645	3.3525	-13.924
-6.07	22.09	-4.12	37.92	1.95	15.83	0.415	0.061212	11.43472	6.13121	-10.655
-10.85	21.15	-10.24	24.59	0.61	3.44	0.415	-2.48324	15.85666	8.36676	-5.2933
-14.56	19.58	-15.64	20.21	-1.08	0.63	0.415	-5.95545	18.0534	8.60455	-1.5266
-17.95	15.04	-18.26	18.01	-0.31	2.97	0.415	-9.52634	18.68694	8.42366	3.64694
0	0	-21.35	10.49	0	0	0.415	-13.0222	17.17346	0	0
Erms				1.43983	11.0272				7.26402	8.37914
Erms x dan y				7.863596					7.841431	

Tabel B.2 Perhitungan alfa pada video 3

		kalman filter				exponential smoothing				
x	y	predict x	predict y	error x	error y	alfa	predict x	predict y	error x	error y
5.97	7.2	0	0	0	0	0.577				
5.51	13.16	0.54	0.65	-4.97	-12.51	0.577	3.44469	4.1544	-2.0653	-9.0056
4.74	19.27	10.47	25.66	5.73	6.39	0.577	4.636374	9.350631	-0.1036	-9.9194
3.19	21.46	3.96	25.38	0.77	3.92	0.577	4.696166	15.07411	1.50617	-6.3859
0.41	18.64	1.65	23.65	1.24	5.01	0.577	3.827108	18.75877	3.41711	0.11877
-3.75	12.21	-2.36	15.82	1.39	3.61	0.577	1.855437	18.69024	5.60544	6.48024
0	0	-7.92	5.79	0	0	0.577	-1.3789	14.95114	0	0
Erms				3.50988	7.08281				3.15095	7.24268
Erms x dan y				5.589519					5.585018	

Tabel B.3 Perhitungan alfa pada video 4

		kalman filter				exponential smoothing				
x	y	predict x	predict y	error x	error y	alfa	predict x	predict y	error x	error y
2.88	15.82	0	0	0	0	0.357				
0.11	18.48	0.26	1.44	0.15	-17.04	0.357	1.02816	5.64774	0.91816	-12.832
-5.3	20.36	-0.05	35.53	5.25	15.17	0.357	0.700377	10.22886	6.00038	-10.131
-8.08	20.36	-10.7	22.24	-2.62	1.88	0.357	-1.44176	13.84567	6.63824	-6.5143
-14.71	18.01	-10.85	20.36	3.86	2.35	0.357	-3.81161	16.17129	10.8984	-1.8387
-16.56	15.82	-21.35	15.66	-4.79	-0.16	0.357	-7.70234	16.82771	8.85766	1.00771
0	0	-18.42	13.63	0	0	0.357	-10.8645	16.46796	0	0
Erms				3.80245	10.2915				7.45851	7.92641
Erms x dan y				7.758003					7.696016	

Tabel B.4 Perhitungan alfa pada video 5

		kalman filter				exponential smoothing				
x	y	predict x	predict y	error x	error y	alfa	predict x	predict y	error x	error y
4.58	12.37	0	0	0	0	0.436				
2.73	16.45	0.42	1.12	-2.31	-15.33	0.436	1.99688	5.39332	-0.7331	-11.057
-0.05	20.99	5.04	31.77	5.09	10.78	0.436	2.31652	10.21403	2.36652	-10.776
-1.75	21.77	-2.83	25.53	-1.08	3.76	0.436	1.284717	14.91235	3.03472	-6.8576
-3.75	20.99	-3.44	22.56	0.31	1.57	0.436	-0.03842	17.90229	3.71158	-3.0877
-7.61	17.54	-5.76	20.21	1.85	2.67	0.436	-1.65667	19.24853	5.95333	1.70853
-9.93	14.25	-11.47	14.1	-1.54	-0.15	0.436	-4.25232	18.50361	5.67768	4.25361
0	0	-12.24	10.96	0	0	0.436	-6.72779	16.64904	0	0
Erms				2.52655	7.90539				4.01666	7.25655
Erms x dan y				5.868504					5.864772	

Tabel B.5 Perhitungan alfa pada video 6

		kalman filter				exponential smoothing				
x	y	predict x	predict y	error x	error y	alfa	predict x	predict y	error x	error y
2.42	13.94	0	0	0	0	0.425				
-0.36	17.23	0.22	1.27	0.58	-15.96	0.425	1.0285	5.9245	1.3885	-11.306
-5.14	20.21	-0.94	33.19	4.2	12.98	0.425	0.438388	10.72934	5.57839	-9.4807
-8.08	20.52	-9.93	23.18	-1.85	2.66	0.425	-1.93243	14.75862	6.14757	-5.7614
-11.32	20.05	-11.01	20.83	0.31	0.78	0.425	-4.54515	17.20721	6.77485	-2.8428
-16.56	17.23	-14.56	19.58	2	2.35	0.425	-7.42446	18.41539	9.13554	1.18539
0	0	-21.81	14.41	0	0	0.425	-11.3071	17.9116	0	0
Erms				2.2581	9.34247				6.32765	7.21633
Erms x dan y				6.796348					6.786554	

Tabel B.6 Perhitungan alfa pada video 7

		kalman filter				exponential smoothing				
x	y	predict x	predict y	error x	error y	alfa	predict x	predict y	error x	error y
6.43	8.61	0	0	0	0	0.6285				
1.19	14.41	0.58	0.78	-0.61	-13.63	0.6285	4.041255	5.411385	2.85126	-8.9986
-4.22	17.54	1.79	28.03	6.01	10.49	0.6285	2.249241	11.06701	6.46924	-6.473
-10.08	17.7	-9.62	20.68	0.46	2.98	0.6285	-1.81668	15.13529	8.26332	-2.5647
-14.25	16.29	-15.95	17.86	-1.7	1.57	0.6285	-7.01018	16.74721	7.23982	0.45721
-17.64	12.53	-18.42	14.88	-0.78	2.35	0.6285	-11.5604	16.45985	6.07959	3.92985
0	0	-21.04	8.77	0	0	0.6285	-15.3814	13.98994	0	0
Erms				2.83557	7.90803				6.44422	5.38712
Erms x dan y				5.940429					5.939233	

Tabel B.7 Perhitungan alfa pada video 8

		kalman filter				exponential smoothing				
x	y	predict x	predict y	error x	error y	alfa	predict x	predict y	error x	error y
1.03	8.3	0	0	0	0	0.435				
-0.82	16.6	0.09	0.75	0.91	-15.85	0.435	0.44805	3.6105	1.26805	-12.99
-1.9	20.21	-1.74	32.45	0.16	12.24	0.435	-0.10355	9.260933	1.79645	-10.949
-3.44	20.52	-2.98	23.81	0.46	3.29	0.435	-0.88501	14.02378	2.55499	-6.4962
-4.53	19.58	-4.99	20.83	-0.46	1.25	0.435	-1.99643	16.84963	2.53357	-2.7304
-5.91	17.7	-5.61	18.64	0.3	0.94	0.435	-3.09853	18.03734	2.81147	0.33734
-9.77	10.65	-7.3	15.82	2.47	5.17	0.435	-4.32152	17.8906	5.44848	7.2406
0	0	-13.63	3.6	0	0	0.435	-6.69161	14.74094	0	0
Erms				1.11563	8.5736				3.03788	8.07057
Erms x dan y				6.113557					6.09766	

Tabel B.8 Perhitungan alfa pada video 9

		kalman filter				exponential smoothing				
x	y	predict x	predict y	error x	error y	alfa	predict x	predict y	error x	error y
1.34	12.84	0	0	0	0	0.79				
0.26	15.35	2.68	25.68	2.42	10.33	0.79	1.0586	10.1436	0.7986	-5.2064
-1.9	17.7	-0.82	17.86	1.08	0.16	0.79	0.427706	14.25666	2.32771	-3.4433
-3.44	17.7	-4.06	20.05	-0.62	2.35	0.79	-1.41118	16.9769	2.02882	-0.7231
-7.92	15.66	-4.99	17.7	2.93	2.04	0.79	-3.01395	17.54815	4.90605	1.88815
-10.7	13.63	-12.4	13.63	-1.7	0	0.79	-6.88973	16.05651	3.81027	2.42651
-15.02	10.33	-13.48	11.59	1.54	1.26	0.79	-9.89984	14.13957	5.12016	3.80957
0	0	-19.34	7.04	0	0	0.79	-13.9448	11.13001	0	0
Erms				1.8821	4.43483				3.53496	3.25197
Erms x dan y				3.40661					3.396414	

Tabel B.9 Perhitungan alfa pada video 10

		kalman filter				exponential smoothing				
x	y	predict x	predict y	error x	error y	alfa	predict x	predict y	error x	error y
6.43	7.04	0	0	0	0	0.527				
7.21	13.16	0.58	0.64	-6.63	-12.52	0.527	3.38861	3.71008	-3.8214	-9.4499
7.21	15.51	13.83	25.67	6.62	10.16	0.527	5.402483	8.690188	-1.8075	-6.8198
6.12	18.17	7.21	17.86	1.09	-0.31	0.527	6.355044	12.28423	0.23504	-5.8858
4.43	17.39	5.04	20.83	0.61	3.44	0.527	6.231176	15.38603	1.80118	-2.004
1.03	14.88	2.73	16.6	1.7	1.72	0.527	5.281956	16.44212	4.25196	1.56212
-3.29	9.39	-2.36	12.37	0.93	2.98	0.527	3.041175	15.61888	6.33118	6.22888
-6.07	5.16	-7.61	3.91	-1.54	-1.25	0.527	-0.29535	12.33626	5.77465	7.17626
0	0	-8.85	0.93	0	0	0.527	-3.33859	8.554372	0	0
Erms				3.693	6.38421				4.01211	6.17838
Erms x dan y				5.215191					5.2091	

c. Data Koordinat 3 Dimensi Sebenarnya dan Prediksi Exponential Smoothing

C.1 Data Koordinat 3 Dimensi Sebenarnya dan Prediksi Exponential Smoothing Video 2

sebenarnya			expoential smoothing					
z	x	y	predict x	predict y	predict z	eror x	eror y	error z
172.66	3.5	13.63	0	0	160			
166.76	-1.9	19.58	1.74843	6.80887	166.324	3.6484	-12.77	-0.4357
155.87	-6.07	22.09	-0.0741	13.1887	166.542	5.9959	-8.901	10.67196
161.17	-10.9	21.15	-3.0694	17.6353	161.211	7.7806	-3.515	0.04078
155.87	-14.6	19.58	-6.9562	19.3911	161.19	7.6038	-0.189	5.320408
147.82	-18	15.04	-10.755	19.4855	158.533	7.1953	4.4455	10.7126
97.78			-14.349	17.2647	153.181	0	0	
Erms						6.624	7.4093	7.171452
Erms x, y, z						7.0759		

C.2 Data Koordinat 3 Dimensi Sebenarnya dan Prediksi Exponential Smoothing Video 3

sebenarnya			expoential smoothing					
z	x	y	predict x	predict y	predict z	eror x	eror y	error z
170.41	5.97	7.2	0	0	160			
165.33	5.51	13.16	2.98231	3.59676	165.2	-2.528	-9.563	-0.12968
163.23	4.74	19.27	4.24502	8.37408	165.265	-0.495	-10.9	2.035099
163.93	3.19	21.46	4.49229	13.8171	164.248	1.3023	-7.643	0.318465
164.63	0.41	18.64	3.84173	17.6351	164.089	3.4317	-1.005	-0.54062
175.73	-3.75	12.21	2.12741	18.1371	164.359	5.8774	5.9271	-11.3706
45.64			-0.8087	15.1762	170.04	0	0	
Erms						3.3061	7.8068	5.173812
Erms x, y, z						5.7342		

C.3 Data Koordinat 3 Dimensi Sebenarnya dan Prediksi Exponential Smoothing Video 4

sebenarnya			expoential smoothing					
z	x	y	predict x	predict y	predict z	eror x	eror y	error z
173.42	2.88	15.82	0	0	160			
157.82	0.11	18.48	1.4387	7.90288	166.704	1.3287	-10.58	8.883961
171.15	-5.3	20.36	0.77495	13.1867	162.266	6.0749	-7.173	-8.88402
160.49	-8.08	20.36	-2.2598	16.7701	166.704	5.8202	-3.59	6.213991
167.48	-14.7	18.01	-5.1673	18.5634	163.6	9.5427	0.5534	-3.88021
152.69	-16.6	15.82	-9.9343	18.287	165.538	6.6257	2.467	12.84815
45.64			-13.244	17.0546	159.12	0	0	
Erms						6.4422	6.0434	8.678655
Erms x, y, z						7.1494		

C.4 Data Koordinat 3 Dimensi Sebenarnya dan Prediksi Exponential Smoothing Video 5

sebenarnya			expoential smoothing					
z	x	y	predict x	predict y	predict z	eror x	eror y	error z
173.42	4.58	12.37	0	0	160			
174.95	2.73	16.45	2.28794	6.17943	166.704	-0.442	-10.27	-8.24604
165.33	-0.05	20.99	2.50877	11.3101	170.823	2.5588	-9.68	5.49327
157.82	-1.75	21.77	1.23054	16.1457	168.079	2.9805	-5.624	10.25911
155.23	-3.75	20.99	-0.2584	18.9553	162.954	3.4916	-2.035	7.72417
167.48	-7.61	17.54	-2.0026	19.9717	159.096	5.6074	2.4317	-8.38444
166.76	-9.93	14.25	-4.8038	18.757	163.284	5.1262	4.507	-3.47599
45.64			-7.3646	16.5055	165.02	0	0	
Erms						3.7758	6.5978	7.449422
Erms x, y, z						6.1449		

C.5 Data Koordinat 3 Dimensi Sebenarnya dan Prediksi Exponential Smoothing Video 6

sebenarnya			expoential smoothing					
z	x	y	predict x	predict y	predict z	eror x	eror y	error z
176.51	2.42	13.94	0	0	160			
159.82	-0.36	17.23	2.42	13.94	176.51	2.78	-3.29	16.69
169.67	-5.14	20.21	1.03125	15.5835	168.173	6.1713	-4.626	-1.49749
161.17	-8.08	20.52	-2.0516	17.8947	168.921	6.0284	-2.625	7.750581
153.95	-11.3	20.05	-5.0631	19.2062	165.049	6.2569	-0.844	11.09878
164.63	-16.6	17.23	-8.1887	19.6277	159.504	8.3713	2.3977	-5.12562
45.64			-12.371	18.4299	162.065	0	0	
Erms						6.1868	3.0193	9.902777
Erms x, y, z						6.9632		

C.6 Data Koordinat 3 Dimensi Sebenarnya dan Prediksi Exponential Smoothing Video 7

sebenarnya			expoential smoothing					
z	x	y	predict x	predict y	predict z	eror x	eror y	error z
164.63	6.43	8.61	0	0	160			
165.33	1.19	14.41	3.21211	4.30113	162.313	2.0221	-10.11	-3.01708
163.23	-4.22	17.54	2.20196	9.35101	163.82	6.422	-8.189	0.590101
165.33	-10.1	17.7	-1.0061	13.4418	163.525	9.0739	-4.258	-1.80468
172.66	-14.3	16.29	-5.539	15.569	164.427	8.711	-0.721	-8.23315
163.23	-17.6	12.53	-9.8906	15.9292	168.54	7.7494	3.3992	5.309718
			-13.762	14.2311	165.887	0	0	
Erms						7.2609	6.3159	4.662313
Erms x, y, z						6.1738		

C.7 Data Koordinat 3 Dimensi Sebenarnya dan Prediksi Exponential Smoothing Video 8

sebenarnya			expoential smoothing					
z	x	y	predict x	predict y	predict z	eror x	eror y	error z
187.17	1.03	8.3	0	0	160			
165.33	-0.82	16.6	0.51454	4.14627	173.573	1.3345	-12.45	8.242773
161.17	-1.9	20.21	-0.1521	10.3675	169.455	1.7479	-9.842	8.285096
160.49	-3.44	20.52	-1.0253	15.2843	165.316	2.4147	-5.236	4.826276
159.82	-4.53	19.58	-2.2316	17.8998	162.905	2.2984	-1.68	3.08531
159.82	-5.91	17.7	-3.3797	18.7391	161.364	2.5303	1.0391	1.544043
169.67	-9.77	10.65	-4.6437	18.22	160.593	5.1263	7.57	-9.07728
			-7.2046	14.4384	165.127	0	0	
Erms						2.8469	7.5343	6.103049
Erms x, y, z						5.8343		

C.8 Data Koordinat 3 Dimensi Sebenarnya dan Prediksi Exponential Smoothing Video 9

sebenarnya			expoential smoothing					
z	x	y	predict x	predict y	predict z	eror x	eror y	error z
174.18	1.34	12.84	0	0	160			
159.15	0.26	15.35	0.6694	6.41422	167.084	0.4094	-8.936	7.933619
163.23	-1.9	17.7	0.46488	10.8781	163.12	2.3649	-6.822	-0.10962
158.48	-3.44	17.7	-0.7165	14.286	163.175	2.7235	-3.414	4.69514
164.63	-7.92	15.66	-2.077	15.9915	160.83	5.843	0.3315	-3.80032
159.15	-10.7	13.63	-4.9959	15.8259	162.728	5.7041	2.1959	3.578131
167.48	-15	10.33	-7.8454	14.7289	160.941	7.1746	4.3989	-6.53932
			-11.429	12.5314	164.207	0	0	
Erms						4.6785	5.2014	4.291027
Erms x, y, z						4.7383		

C.9 Data Koordinat 3 Dimensi Sebenarnya dan Prediksi Exponential Smoothing Video 10

sebenarnya			expoential smoothing					
z	x	y	predict x	predict y	predict z	eror x	eror y	error z
168.93	6.43	7.04	0	0	160			
161.85	7.21	13.16	3.21211	3.51683	164.461	-3.998	-9.643	2.610982
159.15	7.21	15.51	5.20925	8.33408	163.157	-2.001	-7.176	4.006666
160.49	6.12	18.17	6.20873	11.9188	161.155	0.0887	-6.251	0.665136
161.85	4.43	17.39	6.1644	15.0416	160.823	1.7344	-2.348	-1.02713
161.85	1.03	14.88	5.29798	16.2147	161.336	4.268	1.3347	-0.51403
176.51	-3.29	9.39	3.16591	15.548	161.593	6.4559	6.158	-14.9172
178.88	-6.07	5.16	-0.0591	12.4718	169.045	6.0109	7.3118	-9.83534
			-3.0619	8.81917	173.958	0	0	
Erms						4.1236	6.3498	8.012736
Erms x, y, z						6.3647		

D. Kode program

D. 1 Kode Program load video dan prediksi koordinat x dan y filter kalman

```
#include "cv.h"
#include "cxmisc.h"
#include "highgui.h"
#include <vector>
#include <string>
#include <algorithm>
#include <stdio.h>
#include <ctype.h>
#include <iostream>

#define PI 3.14159265

using namespace std;
vector<CvPoint> mousev, kalmanv;
char datasend[8] = { '>','0','0','0','0','0','0','#' };
CvPoint mousep[7];
CvPoint statpos[17];
CvPoint MouseFilter(CvPoint mop);
int sta_test(CvPoint pos);

CvPoint pt = cvPoint(0, 0);
int add_pt = 0;

CvPoint g_XYkiri, g_XYkanan;


using namespace std;

static void prosedurKalibrasi(const char* imageList,
int nx, int ny, int useUncalibrated, float
_squareSize);
void prosedurCapture();
void mencariPosisiBlob(IplImage* src, IplImage* dst,
CvPoint *pete);
```

```
void programFilterDanRemap();
void kalmanFilter(char* windowName, CvPoint center,
IplImage* tampak);
float mapping(float x, float in_min, float in_max,
float out_min, float out_max);
bool pilihanVid = 0;

int main(int argc, char** argv)
{while (1) {
		char Ta;
		cout << "Apakah anda ingin mengambil
gambar lagi \nuntuk kalibarasi? (Y/n)\n";
		cin >> Ta;
		if ((Ta == 'Y') || (Ta == 'y')) {
			prosedurCapture();
			break;
		}
		else if (Ta == 'n') break;
		else printf("Masukkan karakter Y atau
n\n\n");
	}
	while (1) {
		char Ta;
		cout << "Apakah anda ingin kalibarasi?
(Y/n)\n";
		cin >> Ta;
		if ((Ta == 'Y') || (Ta == 'y')) {

	prosedurKalibrasi("stereo_calib.txt", 9, 6, 0,
2.4);			break;
		}
		else if (Ta == 'n') break;
		else printf("Masukkan karakter Y atau
n\n\n");
	}
	while (1) {
		char Ta;
		cout << "Apakah anda ingin load dari
video? (Y/n)\n";
		cin >> Ta;
```

```
                if ((Ta == 'Y') || (Ta == 'y')) {
                        pilihanVid = 1;
                        break;
                }
                else if (Ta == 'n') break;
                else printf("Masukkan karakter Y atau
n\n\n");
        }
        programFilterDanRemap();
        return 0;
}
```

Kalibrasi

```
/* This is inspired from the OpenCV book*/
void prosedurCapture() {
        cvNamedWindow("Kalib kiri", 1);
        cvNamedWindow("Kalib kanan", 1);
        string namaFile1 = "", namaFile2 = "";
        bool find1, find2;
        vector<CvPoint2D32f> temp;
        temp.resize(9 * 6); //ojo lali
        int count = 0;
        int angkafile = 1;
        IplImage *frame1, *frame2;
        CvCapture* capture1 = cvCaptureFromCAM(1);
        // menentukan device untuk mengcapture
gambar(kamera),0 merupakan ID device
        cvSetCaptureProperty(capture1,
CV_CAP_PROP_FRAME_WIDTH, 320);
        cvSetCaptureProperty(capture1,
CV_CAP_PROP_FRAME_HEIGHT, 240);
        CvCapture* capture2 = cvCaptureFromCAM(2);
        cvSetCaptureProperty(capture2,
CV_CAP_PROP_FRAME_WIDTH, 320);
        cvSetCaptureProperty(capture2,
CV_CAP_PROP_FRAME_HEIGHT, 240);
        for (;;) {
                frame1 = cvQueryFrame(capture1);
                frame2 = cvQueryFrame(capture2);
```

```
                if (!frame1)printf("cannot capture
kiri\n");
                if (!frame2)printf("cannot capture
kanan\n");
                IplImage *gray1 =
cvCreateImage(cvGetSize(frame1), 8, 1);
                IplImage *gray2 =
cvCreateImage(cvGetSize(frame2), 8, 1);
                cvCvtColor(frame1, gray1, CV_BGR2GRAY);
                cvCvtColor(frame2, gray2, CV_BGR2GRAY);
                ///nx=9 dan ny=6, ini adalah jumlah kotak
papan catur
                find1 = cvFindChessboardCorners(gray1,
cvSize(/*nx*/9, /*ny*/6),
                        &temp[0], &count,
CV_CALIB_CB_ADAPTIVE_THRESH |
CV_CALIB_CB_NORMALIZE_IMAGE);
                find2 = cvFindChessboardCorners(gray2,
cvSize(/*nx*/9, /*ny*/6),
                        &temp[0], &count,
CV_CALIB_CB_ADAPTIVE_THRESH |
CV_CALIB_CB_NORMALIZE_IMAGE);
                char c = cvWaitKey(99);
                if (find1 && find2) {
                        cvCircle(gray1, cvPoint(0, 0), 10,
CV_RGB(0, 255, 0), CV_FILLED);
                        cvCircle(gray2, cvPoint(0, 0), 10,
CV_RGB(0, 255, 0), CV_FILLED);
                }
                if (find1 && find2 && c == ' ') {
                        stringstream bufferString1,
bufferString2;
                        if (angkafile <10) {
                                bufferString1 << "left" <<
"0" << angkafile << ".jpg";
                                bufferString2 << "right" <<
"0" << angkafile << ".jpg";
                        }
                        else {
        bufferString1 << "left" << angkafile << ".jpg";
```

```
        bufferString2 << "right" << angkafile << ".jpg";
                        }
                        namaFile1 = bufferString1.str();
                        namaFile2 = bufferString2.str();
                        cvSaveImage(namaFile1.c_str(),
gray1);
                        cvSaveImage(namaFile2.c_str(),
gray2);
                        printf("lalala %d \n", angkafile);
                        angkafile++;
                }
                cvShowImage("Kalib kiri", gray1);
                cvShowImage("Kalib kanan", gray2);
                cvReleaseImage(&gray1);
                cvReleaseImage(&gray2);
                if (angkafile == 20) break; //jumlah
gambar yg di capture
        }
        cvDestroyAllWindows();
        cvReleaseImage(&frame1);
        cvReleaseImage(&frame2);
        cvReleaseCapture(&capture1);
        cvReleaseCapture(&capture2);
}

/* by Martin Peris Martorell (info@martinperis.com)*/
static void prosedurKalibrasi(const char* imageList,
int nx, int ny, int useUncalibrated, float _squareSize)
{
        int displayCorners = 1;
        int showUndistorted = 1;
        bool isVerticalStereo = false;//OpenCV can handle
left-right

//or up-down camera arrangements
        const int maxScale = 1;
        const float squareSize = _squareSize;
//Chessboard square size in cm

        FILE* f = fopen(imageList, "rt");
```

```
	int i, j, lr, nframes, n = nx*ny, N = 0;
	vector<string> imageNames[2];
	vector<CvPoint3D32f> objectPoints;
	vector<CvPoint2D32f> points[2];
	vector<int> npoints;
	vector<uchar> active[2];
	vector<CvPoint2D32f> temp(n);
	CvSize imageSize = { 0,0 };
	// ARRAY AND VECTOR STORAGE:
	double M1[3][3], M2[3][3], D1[5], D2[5];
	double R[3][3], T[3], E[3][3], F[3][3];
	double Q[4][4];
	CvMat _M1 = cvMat(3, 3, CV_64F, M1);
	CvMat _M2 = cvMat(3, 3, CV_64F, M2);
	CvMat _D1 = cvMat(1, 5, CV_64F, D1);
	CvMat _D2 = cvMat(1, 5, CV_64F, D2);
	CvMat _R = cvMat(3, 3, CV_64F, R);
	CvMat _T = cvMat(3, 1, CV_64F, T);
	CvMat _E = cvMat(3, 3, CV_64F, E);
	CvMat _F = cvMat(3, 3, CV_64F, F);
	CvMat _Q = cvMat(4, 4, CV_64F, Q);
	if (displayCorners)
		cvNamedWindow("corners", 1);
	// READ IN THE LIST OF CHESSBOARDS:
	if (!f)
	{
		fprintf(stderr, "can not open file %s\n",
imageList);
		return;
	}
	for (i = 0;;i++)
	{
		char buf[1024];
		int count = 0, result = 0;
		lr = i % 2;
		vector<CvPoint2D32f>& pts = points[lr];
		if (!fgets(buf, sizeof(buf) - 3, f))
			break;
		size_t len = strlen(buf);
```

```
		while (len > 0 && isspace(buf[len - 1]))
			buf[--len] = '\0';
		if (buf[0] == '#')
			continue;
		IplImage* img = cvLoadImage(buf, 0);
		if (!img)
			break;
		imageSize = cvGetSize(img);
		imageNames[lr].push_back(buf);
		//FIND CHESSBOARDS AND CORNERS THEREIN:
		for (int s = 1; s <= maxScale; s++)
		{
			IplImage* timg = img;
			if (s > 1)
			{
				timg =
cvCreateImage(cvSize(img->width*s, img->height*s),
					img->depth, img-
>nChannels);
				cvResize(img, timg,
CV_INTER_CUBIC);
			}
			result =
cvFindChessboardCorners(timg, cvSize(nx, ny),
				&temp[0], &count,
				CV_CALIB_CB_ADAPTIVE_THRESH
|

	CV_CALIB_CB_NORMALIZE_IMAGE);
			if (timg != img)
				cvReleaseImage(&timg);
			if (result || s == maxScale)
				for (j = 0; j < count; j++)
				{
					temp[j].x /= s;
					temp[j].y /= s;
				}
			if (result)
				break;
		}
```

```
                if (displayCorners)
                {
                        printf("%s\n", buf);
                        IplImage* cimg =
cvCreateImage(imageSize, 8, 3);
                        cvCvtColor(img, cimg, CV_GRAY2BGR);
                        cvDrawChessboardCorners(cimg,
cvSize(nx, ny), &temp[0],
                                count, result);
                        cvShowImage("corners", cimg);
                        cvReleaseImage(&cimg);
                        if (cvWaitKey(0) == 27) //Allow ESC
to quit
                                exit(-1);
                }
                else
                        putchar('.');
                N = pts.size();
                pts.resize(N + n, cvPoint2D32f(0, 0));
                active[lr].push_back((uchar)result);
                //assert( result != 0 );
                if (result)
                {
                        //Calibration will suffer without
subpixel interpolation
                        cvFindCornerSubPix(img, &temp[0],
count,
                                cvSize(11, 11), cvSize(-1, -
1),

        cvTermCriteria(CV_TERMCRIT_ITER +
CV_TERMCRIT_EPS,
                                        30, 0.01));
                        copy(temp.begin(), temp.end(),
pts.begin() + N);
                }
                cvReleaseImage(&img);
        }
        fclose(f);
        printf("\n");
```

```
	// HARVEST CHESSBOARD 3D OBJECT POINT LIST:
	nframes = active[0].size();//Number of good
chessboads found
	objectPoints.resize(nframes*n);
	for (i = 0; i < ny; i++)
		for (j = 0; j < nx; j++)
			objectPoints[i*nx + j] =
cvPoint3D32f(i*squareSize, j*squareSize, 0);
	for (i = 1; i < nframes; i++)
		copy(objectPoints.begin(),
objectPoints.begin() + n,
			objectPoints.begin() + i*n);
	npoints.resize(nframes, n);
	N = nframes*n;
	CvMat _objectPoints = cvMat(1, N, CV_32FC3,
&objectPoints[0]);
	CvMat _imagePoints1 = cvMat(1, N, CV_32FC2,
&points[0][0]);
	CvMat _imagePoints2 = cvMat(1, N, CV_32FC2,
&points[1][0]);
	CvMat _npoints = cvMat(1, npoints.size(), CV_32S,
&npoints[0]);
	cvSetIdentity(&_M1);
	cvSetIdentity(&_M2);
	cvZero(&_D1);
	cvZero(&_D2);

	// CALIBRATE THE STEREO CAMERAS
	printf("Running stereo calibration ...");
	fflush(stdout);
	cvStereoCalibrate(&_objectPoints, &_imagePoints1,
		&_imagePoints2, &_npoints,
		&_M1, &_D1, &_M2, &_D2,
		imageSize, &_R, &_T, &_E, &_F,
		cvTermCriteria(CV_TERMCRIT_ITER +
			CV_TERMCRIT_EPS, 100, 1e-5),
		CV_CALIB_FIX_ASPECT_RATIO +
		CV_CALIB_ZERO_TANGENT_DIST +
		CV_CALIB_SAME_FOCAL_LENGTH);
	printf(" done\n");
```

```
	// CALIBRATION QUALITY CHECK
	// because the output fundamental matrix
implicitly
	// includes all the output information,
	// we can check the quality of calibration using
the
	// epipolar geometry constraint: m2^t*F*m1=0
	vector<CvPoint3D32f> lines[2];
	points[0].resize(N);
	points[1].resize(N);
	_imagePoints1 = cvMat(1, N, CV_32FC2,
&points[0][0]);
	_imagePoints2 = cvMat(1, N, CV_32FC2,
&points[1][0]);
	lines[0].resize(N);
	lines[1].resize(N);
	CvMat _L1 = cvMat(1, N, CV_32FC3, &lines[0][0]);
	CvMat _L2 = cvMat(1, N, CV_32FC3, &lines[1][0]);
	//Always work in undistorted space
	cvUndistortPoints(&_imagePoints1, &_imagePoints1,
		&_M1, &_D1, 0, &_M1);
	cvUndistortPoints(&_imagePoints2, &_imagePoints2,
		&_M2, &_D2, 0, &_M2);
	cvComputeCorrespondEpilines(&_imagePoints1, 1,
&_F, &_L1);
	cvComputeCorrespondEpilines(&_imagePoints2, 2,
&_F, &_L2);
	double avgErr = 0;
	for (i = 0; i < N; i++)
	{
		double err =
fabs(points[0][i].x*lines[1][i].x +
			points[0][i].y*lines[1][i].y +
lines[1][i].z)
			+ fabs(points[1][i].x*lines[0][i].x
+
				points[1][i].y*lines[0][i].y
+ lines[0][i].z);
		avgErr += err;
	}
```

```
        printf("avg err = %g\n", avgErr / (nframes*n));
        //COMPUTE AND DISPLAY RECTIFICATION
        if (showUndistorted)
        {
                CvMat* mx1 = cvCreateMat(imageSize.height,
                        imageSize.width, CV_32F);
                CvMat* my1 = cvCreateMat(imageSize.height,
                        imageSize.width, CV_32F);
                CvMat* mx2 = cvCreateMat(imageSize.height,

                        imageSize.width, CV_32F);
                CvMat* my2 = cvCreateMat(imageSize.height,
                        imageSize.width, CV_32F);
                CvMat* img1r =
cvCreateMat(imageSize.height,
                        imageSize.width, CV_8U);
                CvMat* img2r =
cvCreateMat(imageSize.height,
                        imageSize.width, CV_8U);
                CvMat* disp =
cvCreateMat(imageSize.height,
                        imageSize.width, CV_16S);
                CvMat* vdisp =
cvCreateMat(imageSize.height,
                        imageSize.width, CV_8U);
                CvMat* pair;
                double R1[3][3], R2[3][3], P1[3][4],
P2[3][4];
                CvMat _R1 = cvMat(3, 3, CV_64F, R1);
                CvMat _R2 = cvMat(3, 3, CV_64F, R2);
                // IF BY CALIBRATED (BOUGUET'S METHOD)
                if (useUncalibrated == 0)
                {
                        CvMat _P1 = cvMat(3, 4, CV_64F,
P1);
                        CvMat _P2 = cvMat(3, 4, CV_64F,
P2);
                        cvStereoRectify(&_M1, &_M2, &_D1,
&_D2, imageSize,
                                &_R, &_T,
```

```
                        &_R1, &_R2, &_P1, &_P2, &_Q,

      0/*CV_CALIB_ZERO_DISPARITY*/);
                    isVerticalStereo = fabs(P2[1][3]) >
fabs(P2[0][3]);
                    //Precompute maps for cvRemap()
                    cvInitUndistortRectifyMap(&_M1,
&_D1, &_R1, &_P1, mx1, my1);
                    cvInitUndistortRectifyMap(&_M2,
&_D2, &_R2, &_P2, mx2, my2);

                    //Save parameters
                    cvSave("M1.xml", &_M1);
                    cvSave("D1.xml", &_D1);
                    cvSave("R1.xml", &_R1);
                    cvSave("P1.xml", &_P1);
                    cvSave("M2.xml", &_M2);
                    cvSave("D2.xml", &_D2);
                    cvSave("R2.xml", &_R2);
                    cvSave("P2.xml", &_P2);
                    cvSave("Q.xml", &_Q);
                    cvSave("mx1.xml", mx1);
                    cvSave("my1.xml", my1);
                    cvSave("mx2.xml", mx2);
                    cvSave("my2.xml", my2);

            }
            //OR ELSE HARTLEY'S METHOD
            else if (useUncalibrated == 1 ||
useUncalibrated == 2)
                    // use intrinsic parameters of each
camera, but
                    // compute the rectification
transformation directly
                    // from the fundamental matrix
            {
                    double H1[3][3], H2[3][3],
iM[3][3];
                    CvMat _H1 = cvMat(3, 3, CV_64F,
H1);
```

```
                    CvMat _H2 = cvMat(3, 3, CV_64F,
H2);
                    CvMat _iM = cvMat(3, 3, CV_64F,
iM);
                    //Just to show you could have
independently used F
                    if (useUncalibrated == 2)

       cvFindFundamentalMat(&_imagePoints1,
                                  &_imagePoints2, &_F);

       cvStereoRectifyUncalibrated(&_imagePoints1,
                           &_imagePoints2, &_F,
                           imageSize,
                           &_H1, &_H2, 3);
                    cvInvert(&_M1, &_iM);
                    cvMatMul(&_H1, &_M1, &_R1);
                    cvMatMul(&_iM, &_R1, &_R1);
                    cvInvert(&_M2, &_iM);
                    cvMatMul(&_H2, &_M2, &_R2);
                    cvMatMul(&_iM, &_R2, &_R2);
                    //Precompute map for cvRemap()
                    cvInitUndistortRectifyMap(&_M1,
&_D1, &_R1, &_M1, mx1, my1);

                    cvInitUndistortRectifyMap(&_M2,
&_D1, &_R2, &_M2, mx2, my2);
              }
              else
                    assert(0);
              cvNamedWindow("rectified", 1);
              // RECTIFY THE IMAGES AND FIND DISPARITY
MAPS
              if (!isVerticalStereo)
                    pair =
cvCreateMat(imageSize.height, imageSize.width * 2,
                           CV_8UC3);
              else
                    pair = cvCreateMat(imageSize.height
* 2, imageSize.width,
```

```
                        CV_8UC3);
            //Setup for finding stereo
corrrespondences
            CvStereoBMState *BMState =
cvCreateStereoBMState();
            assert(BMState != 0);
            BMState->preFilterSize = 41;
            BMState->preFilterCap = 31;
            BMState->SADWindowSize = 41;
            BMState->minDisparity = -64;
            BMState->numberOfDisparities = 128;
            BMState->textureThreshold = 10;
            BMState->uniquenessRatio = 15;
            for (i = 0; i < nframes; i++)
            {
                  IplImage* img1 =
cvLoadImage(imageNames[0][i].c_str(), 0);
                  IplImage* img2 =
cvLoadImage(imageNames[1][i].c_str(), 0);
                  if (img1 && img2)
                  {
                        CvMat part;
                        cvRemap(img1, img1r, mx1,
my1);
                        cvRemap(img2, img2r, mx2,
my2);
                        if (!isVerticalStereo ||
useUncalibrated != 0)
                        {


      cvFindStereoCorrespondenceBM(img1r, img2r, disp,
                                    BMState);
      cvNormalize(disp, vdisp, 0, 256, CV_MINMAX);

      cvNamedWindow("disparity");
      cvShowImage("disparity", vdisp);
                        }
                        if (!isVerticalStereo)
                        {
```

```
        cvGetCols(pair, &part, 0, imageSize.width);
        cvCvtColor(img1r, &part, CV_GRAY2BGR);
        cvGetCols(pair, &part, imageSize.width,
        imageSize.width * 2);
        cvCvtColor(img2r, &part, CV_GRAY2BGR);
        for (j = 0; j < imageSize.height; j += 16)
        cvLine(pair, cvPoint(0, j);
        cvPoint(imageSize.width * 2, j)
                        CV_RGB(0, 255, 0));}
                                else
{cvGetRows(pair, &part, 0, imageSize.height);
cvCvtColor(img1r, &part, CV_GRAY2BGR);
cvGetRows(pair, &part, imageSize.height,
imageSize.height * 2);
cvCvtColor(img2r, &part, CV_GRAY2BGR);
for (j = 0; j < imageSize.width; j += 16)
                        cvLine(pair, cvPoint(j, 0),

        cvPoint(j, imageSize.height * 2),
        CV_RGB(0, 255, 0));}
                        cvShowImage("rectified", pair);
                                if (cvWaitKey() == 27)
                                        break;}
                        cvReleaseImage(&img1);
                        cvReleaseImage(&img2);}
                cvReleaseStereoBMState(&BMState);
                cvReleaseMat(&mx1);
                cvReleaseMat(&my1);
                cvReleaseMat(&mx2);
                cvReleaseMat(&my2);
                cvReleaseMat(&img1r);
                cvReleaseMat(&img2r);
                cvReleaseMat(&disp);}}
```

Menentukan titik tengah

```
void mencariPosisiBlob(IplImage* src, IplImage* dst,
CvPoint *pete) {
        CvMoments* momen =
(CvMoments*)malloc(sizeof(CvMoments));
        cvMoments(src, momen, 1);
        double momen10 = cvGetSpatialMoment(momen, 1, 0);
```

```
        double momen01 = cvGetSpatialMoment(momen, 0, 1);
        double area = cvGetCentralMoment(momen, 0, 0);
        static int posX = 0;
        static int posY = 0;
        int lastX = posX;
        int lastY = posY;
        posX = momen10 / area;
        posY = momen01 / area;
        pete->x = posX;
        pete->y = posY;
        cvCircle(src,
        cvPoint(posX, posY), 3,CV_RGB(255, 0,
0)CV_FILLED);}

void programFilterDanRemap() {
        //char data;
        int x = 0, y = 0;
        int xs = 0, ys = 0, z = 0;
        double param;
        int teta1, teta2;
        param = 1.0;
        int hl = 23, sl = 96, vl = 180, hh = 45, sh =
255, vh = 255;
        int data = 0, H = 0, S = 0, V = 0;
        int ero1 = 1, ero2 = 0, dil1 = 2, dil2 = 0;
        int data2 = 0, Hr = 0, Sr = 0, Vr = 0;
        int ero1r = 1, ero2r = 0, dil1r = 2, dil2r = 0;

        cvNamedWindow("Input kiri", 1);
        cvNamedWindow("Input kanan", 1);
        cvNamedWindow("right result", 1);
        cvNamedWindow("left result", 1);
        cvNamedWindow("trackbar", 1);
        cvNamedWindow("posisi sebenarnya", 1);
        cvResizeWindow("posisi sebenarnya", 500, 500);

        IplImage *frame1, *frame2, *gbKalman;
        gbKalman = cvCreateImage(cvSize(500, 500), 8, 3);
        CvCapture* capture, *capture2;
        if (pilihanVid) {
```

```
                capture = cvCreateFileCapture("kiri.avi");
                capture2 =
cvCreateFileCapture("kanan.avi");
        }
        else {
                capture = cvCaptureFromCAM(1);
                capture2 = cvCaptureFromCAM(2);
        }
        cvSetCaptureProperty(capture,
CV_CAP_PROP_FRAME_WIDTH, 320);
        cvSetCaptureProperty(capture,
CV_CAP_PROP_FRAME_HEIGHT, 240);
        cvSetCaptureProperty(capture2,
CV_CAP_PROP_FRAME_WIDTH, 320);
        cvSetCaptureProperty(capture2,
CV_CAP_PROP_FRAME_HEIGHT, 240);

cvCreateTrackbar("Hlow", "trackbar", &hl, 179, 0);
cvCreateTrackbar("Slow", "trackbar", &sl, 255, 0);
cvCreateTrackbar("Vlow", "trackbar", &vl, 255, 0);
cvCreateTrackbar("HHigh", "trackbar", &hh, 179, 0);
cvCreateTrackbar("SHigh", "trackbar", &sh, 255, 0);
cvCreateTrackbar("VHigh", "trackbar", &vh, 255, 0);

        CvMat* mx1 = (CvMat*)cvLoad("mx1.xml");
        if (!mx1) {
                printf("matrix tidak dapat di load");
                return;}
        CvMat* my1 = (CvMat*)cvLoad("my1.xml");
        CvMat* mx2 = (CvMat*)cvLoad("mx2.xml");
        CvMat* my2 = (CvMat*)cvLoad("my2.xml");
        CvMat* Q = (CvMat*)cvLoad("Q.xml");
        //============matrix kalman================
        CvKalman* kalman = cvCreateKalman(4, 2, 0);
        const CvMat* state = cvCreateMat(4, 1, CV_32FC1);
        CvMat* measurement = cvCreateMat(2, 1, CV_32FC1);
        cvZero(measurement);
        CvMat* processnoise = cvCreateMat(4, 1,
CV_32FC1);
        cvSetIdentity(kalman->transition_matrix);
```

```
	cvSetIdentity(kalman->measurement_matrix);
	kalman->measurement_matrix->data.fl[0] = 1;
	kalman->measurement_matrix->data.fl[5] = 1;
	kalman->process_noise_cov->data.fl[0] = 1;
	kalman->process_noise_cov->data.fl[5] = 1;
	kalman->process_noise_cov->data.fl[10] = 1;
	kalman->process_noise_cov->data.fl[15] = 1;
	cvSetIdentity(kalman->measurement_noise_cov,
cvScalar(1e-1));
	double ticks;
	bool klik_ya = 0, found_ya = 1;
	int notFound;
	//============tulis file, buka file handler=
	time_t tictoc;
	time(&tictoc);
	FILE *fh;

	fh = fopen("simpananPosisi.txt", "a");
	if (!fh) {
		printf("error!");
		exit(1);}
	fprintf(fh, "\n\nini ditulis pada:\n%s\n",
ctime(&tictoc));
		fprintf(fh, "z actual\tx actual\ty
actual\tx predict\ty predict\n");
	//============tulis rekam menjadi AVI=======
	CvSize size =
cvSize((int)cvGetCaptureProperty(capture,
CV_CAP_PROP_FRAME_WIDTH),
		(int)cvGetCaptureProperty(capture,
CV_CAP_PROP_FRAME_HEIGHT));
	CvVideoWriter* writer1 = 0, *writer2 = 0;
	if (!pilihanVid) {
		writer1 = cvCreateVideoWriter("kiri.avi",
CV_FOURCC('M', 'J', 'P', 'G'),
			5, size);
		writer2 = cvCreateVideoWriter("kanan.avi",
CV_FOURCC('M', 'J', 'P', 'G'),
			5, size);
	}
```

```
	//=========================================
	while (1) {
		frame1 = cvQueryFrame(capture);
		frame2 = cvQueryFrame(capture2);
		if (!frame1) { printf("cannot capture
kiri\n"); break; }
		if (!frame2) { printf("cannot capture
kanan\n"); break; }			if
(!pilihanVid) {
			cvWriteFrame(writer1, frame1);
			cvWriteFrame(writer2, frame2);
		}
		CvSize ukuranFrame1 = cvGetSize(frame1);

		IplImage *output1 =
cvCreateImage(cvGetSize(frame1), 8, 1);
		IplImage *output2 =
cvCreateImage(cvGetSize(frame2), 8, 1);

		cvMirror(frame1, frame1, 90);
		cvMirror(frame2, frame2, 90);

		cvCvtColor(frame1, output1, CV_BGR2GRAY);
		cvCvtColor(frame2, output2, CV_BGR2GRAY);

		CvMat* img1r =
cvCreateMat(ukuranFrame1.height, ukuranFrame1.width,
CV_8U);
		CvMat* img2r =
cvCreateMat(ukuranFrame1.height, ukuranFrame1.width,
CV_8U);

		cvRemap(output1, img1r, mx1, my1);
		cvRemap(output2, img2r, mx2, my2);

		cvShowImage("Input kiri", img1r);
		cvShowImage("Input kanan", img2r);

	cvSmooth(frame2, frame2, CV_GAUSSIAN, 9, 9);
	cvSmooth(frame1, frame1, CV_GAUSSIAN, 9, 9);
```

```
		cvCvtColor(frame1, frame1,
CV_BGR2HSV);//rumus convert bgr ke hsv
		cvCvtColor(frame2, frame2, CV_BGR2HSV);

		cvInRangeS(frame1, cvScalar(hl, sl, vl),
cvScalar(hh, sh, vh), output1);
		cvInRangeS(frame2, cvScalar(hl, sl, vl),
cvScalar(hh, sh, vh), output2);
		cvDilate(output1, output1, NULL, 30);
	cvErode(output1, output1, NULL, 20);
		cvDilate(output2, output2, NULL, 30);
		cvErode(output2, output2, NULL, 20);
//putih melebar

	mencariPosisiBlob(output1, output1, &g_XYkiri);
	mencariPosisiBlob(output2, output2, &g_XYkanan);
		cvRemap(output1, img1r, mx1, my1);
		cvRemap(output2, img2r, mx2, my2);
///=============menghitung jarak===============
		float a, b, e;//faktor koreksi
		float x_x, y_y, z_z;
		float f = 5.4692678627033354e+002;
		float d =  g_XYkanan.x - g_XYkiri.x;
		e = abs(300 * f) / d;
	z_z = e - ((-0.0021*(e*e)) + (1.335*e) - 45.642);
		a = g_XYkanan.x * 160 / f;
		x_x = a - ((1.5276 *(a)) - 30.357);
		b = g_XYkanan.y * 160 / f;
		y_y = b + (-1.5356*b) + 28.667;
		printf("x= %.2f  \t y= %.2f  \t z=
%20.2f\n", x_x, y_y, z_z);
	///=============kalman filter==================
		CvPoint p_ptWindowKalman;
		p_ptWindowKalman.x = cvRound(mapping(x_x,
	-40, 40, 500, 0));
		p_ptWindowKalman.y = cvRound(mapping(y_y,
-40, 40, 500, 0));
		cvCircle(gbKalman, p_ptWindowKalman, 1,
CV_RGB(255, 255, 255), CV_FILLED);
```

```
                //cvMirror(gbKalman, gbKalman, 90);
                cvShowImage("posisi sebenarnya",
gbKalman);
                cvZero(gbKalman);
                double precTick = ticks = 0;
                ticks = (double)cvGetTickCount();
                double dT = (ticks - precTick) /
cvGetTickFrequency();
        ///=============simpan posisi dalam file=======
                fprintf(fh, "%.2f\t%.2f\t%.2f\t", z_z,
x_x, y_y);

        ///============================================
                if (found_ya) {
        kalman->transition_matrix->data.fl[2] = dT;
        kalman->transition_matrix->data.fl[7] = dT;
        state = cvKalmanPredict(kalman, 0);
        CvPoint predictPt = cvPoint(cvRound(state-
>data.fl[0]), cvRound(state->data.fl[1]));
fprintf(fh, " %.2f\t", state->data.fl[0]);
fprintf(fh, " %.2f\t", state->data.fl[1]);
        kalmanv.push_back(cvPoint(mapping(predictPt.x, -
40, 40, 500, 0), mapping(predictPt.y, -40, 40, 500,
0)));
        printf("%d,%d\n", predictPt.x, predictPt.y);
        cvCircle(gbKalman, cvPoint(mapping(predictPt.x, -
40, 40, 500, 0), mapping(predictPt.y, -40, 40, 500,
0)), 2, cvScalar(0, 0, 255), 1);
                }

                int notFound = 0;
                if ((g_XYkiri.x < 1) && (g_XYkiri.y < 1)
&& (g_XYkanan.x < 1) && (g_XYkanan.y < 1)) {
                        notFound++;
                        printf("\t\t\t\t%d\n", notFound);
                        printf("\t\t\t\tprediction x =%d
,prediction y =%d\n", (state->data.fl[0]), (state-
>data.fl[1]));
                        if (notFound > 10) {
                                found_ya = 0;
```

```
                        }
                }
                else {
                        notFound = 0;
                        measurement->data.fl[0] = x_x;
                        measurement->data.fl[1] = y_y;
                        if (!found_ya) {
                                kalman->error_cov_pre-
>data.fl[0] = 1;
                                kalman->error_cov_pre-
>data.fl[5] = 1;
                                kalman->error_cov_pre-
>data.fl[10] = 1;
                                kalman->error_cov_pre-
>data.fl[15] = 1;
                                state->data.fl[0] =
measurement->data.fl[0];
                                state->data.fl[1] =
measurement->data.fl[1];
                                state->data.fl[2] = 0;
                                state->data.fl[3] = 0;
                                found_ya = 1;
                        }
                        else {
                                cvKalmanCorrect(kalman,
measurement);
                                                        }
                }
                mousev.push_back(p_ptWindowKalman);
                for (int i = 0; i < mousev.size() - 1;
i++) {
                        cvLine(gbKalman, mousev[i],
mousev[i + 1], cvScalar(255, 255, 0), 1);///blue line
real time
                }
                for (int i = 0; i < kalmanv.size() - 1;
i++) {
                        cvLine(gbKalman, kalmanv[i],
kalmanv[i + 1], cvScalar(0, 255, 0), 1);///green line
kalman
```

```
                }
                fprintf(fh, "\n");

        ///=============================================
                cvShowImage("left result", img1r);
                cvShowImage("right result", img2r);

                char c = cvWaitKey(99);
                if (c == 'c') {
                        mousev.clear();
                        kalmanv.clear();
                }
                if (c == 27)
                break;

                cvSetTrackbarPos("Hlow", "trackbar", hl);
                cvSetTrackbarPos("Slow", "trackbar", sl);
                cvSetTrackbarPos("Vlow", "trackbar", vl);
                cvSetTrackbarPos("HHigh", "trackbar", hh);
                cvSetTrackbarPos("SHigh", "trackbar", sh);
                cvSetTrackbarPos("VHigh", "trackbar", vh);
                cvReleaseImage(&output1);
                cvReleaseImage(&output2);
                cvReleaseMat(&img1r);
                cvReleaseMat(&img2r);
        }
        cvDestroyAllWindows();
        cvReleaseImage(&frame1);
        cvReleaseImage(&frame2);
        cvReleaseImage(&gbKalman);
        cvReleaseCapture(&capture);
        cvReleaseCapture(&capture2);
        cvReleaseVideoWriter(&writer1);
        cvReleaseVideoWriter(&writer2);
        fclose(fh);
        return;
}

float mapping(float x, float in_min, float in_max,
float out_min, float out_max)
```

```
{
	return (x - in_min) * (out_max - out_min) /
(in_max - in_min) + out_min;
}
```

D.2 Kode program exponential smoothing 3 dimensi

```
#include "cv.h"
#include "cxmisc.h"
#include "highgui.h"
#include <vector>
#include <string>
#include <algorithm>
#include "math.h"
#include <cv.h>
#include <highgui.h>
#include <math.h>
#include <iomanip>
#include <stdio.h>
#include <ctype.h>
#include <iostream>
#include <conio.h>

#define PI 3.14159265

using namespace std;;
char datasend[8] = { '>','0','0','0','0','0','0','#' };
CvPoint mousep[7];
CvPoint statpos[17];
CvPoint MouseFilter(CvPoint mop);
CvPoint ukur, ukur1, ukur2, ukur3, tengah = cvPoint(0,
0);
int sta_test(CvPoint pos);

IplConvKernel *element;			IplConvKernel
*element2;
void GetDesktopResolution(int& horizontal, int&
vertical);  float mapping(float x, float in_min, float
in_max, float out_min, float out_max);
CvPoint pt = cvPoint(0, 0);
int add_pt = 0;
```

```
CvPoint g_XYkiri, g_XYkanan;
int mod(int x, int y) {
	return x%y;
}

void on_mouse(int event, int x, int y, int flags, void*
param)		{
	if (event == CV_EVENT_MOUSEMOVE)//menggunakan
mouse dan klik kiri
	{
		pt = cvPoint(x, y);
		add_pt = 1;

	}
}

int main()
{
	int x = 0, y = 0;
	int xs = 0, ys = 0, z = 0;
	double param;
	int teta1, teta2;
	param = 1.0;
	int hl = 23, sl = 96, vl = 145, hh = 45, sh =
255, vh = 255;
	CvCapture* capture = 0;
	IplImage *frame;
	int data = 0, H = 0, S = 0, V = 0;
	capture = cvCaptureFromCAM(1);
	cvNamedWindow("left result", 1);
	int ero1 = 1, ero2 = 0, dil1 = 2, dil2 = 0;
	int hlr = 23, slr = 96, vlr = 145, hhr = 45, shr
= 255, vhr = 255;
	CvCapture* capture2 = 0;;
	IplImage *frame2;
	int data2 = 0, Hr = 0, Sr = 0, Vr = 0;
	capture2 = cvCaptureFromCAM(0);
	cvNamedWindow("right result", 1);
	int ero1r = 1, ero2r = 0, dil1r = 2, dil2r = 0;
	IplImage *gbKalman;
```

```
	//=======================================
	cvNamedWindow("Color", 1);
	cvCreateTrackbar("Hlow", "Color", &hl, 179, 0);
	cvCreateTrackbar("Slow", "Color", &sl, 255, 0);
	cvCreateTrackbar("Vlow", "Color", &vl, 255, 0);
	cvCreateTrackbar("HHigh", "Color", &hh, 179, 0);
	cvCreateTrackbar("SHigh", "Color", &sh, 255, 0);
	cvCreateTrackbar("VHigh", "Color", &vh, 255, 0);
	//=======================================
	cvNamedWindow("Filter", 1);
	cvCreateTrackbar("erode 1", "Filter", &ero1, 20,
0);
	cvCreateTrackbar("dilate 1", "Filter", &dil1, 20,
0);
	cvCreateTrackbar("erode 2", "Filter", &ero2, 20,
0);
	cvCreateTrackbar("dilate 2", "Filter", &dil2, 20,
0);

	//============tulis file, buka file handler=
	time_t tictoc;
	time(&tictoc);
	FILE *fh;

	fh = fopen("simpananPosisi.txt", "a");
	if (!fh) {
		printf("error!");
		exit(1);
	}
	fprintf(fh, "ini ditulis pada\n\t:%s\n",
ctime(&tictoc));
	fprintf(fh, "x actual\ty actual\tx predict\ty
predict\n");

	while (1)
	{
		frame = cvRetrieveFrame(capture);
		frame2 = cvRetrieveFrame(capture2);
			cvMirror(frame, frame, 90);
```

```
            cvMirror(frame2, frame2, 90);
      IplImage *thres = cvCreateImage(cvSize(frame-
>width, frame->height), 8, 1);
      IplImage *thres2 = cvCreateImage(cvSize(frame2-
>width, frame2->height), 8, 1);
      IplImage *HSV = cvCreateImage(cvSize(frame-
>width, frame->height), 8, 3);
      IplImage *HSV2 = cvCreateImage(cvSize(frame2-
>width, frame2->height), 8, 3);
      IplImage *hasil = cvCreateImage(cvSize(frame-
>width, frame->height), 8, 1);
      IplImage *hasil2 = cvCreateImage(cvSize(frame2-
>width, frame2->height), 8, 1);
            cvCvtColor(frame, HSV, CV_BGR2HSV);//rumus
convert bgr ke hsv
            cvCvtColor(frame2, HSV2, CV_BGR2HSV);
            cvInRangeS(HSV, cvScalar(hl, sl, vl),
cvScalar(hh, sh, vh), thres);
            cvInRangeS(HSV2, cvScalar(hl, sl, vl),
cvScalar(hh, sh, vh), thres2);
            element = cvCreateStructuringElementEx(9,
9, 4, 4, CV_SHAPE_RECT, NULL);
            cvErode(thres, thres, element, ero1);
            cvDilate(thres, thres, element, dil1);
            cvErode(thres, thres, element, ero2);
            cvDilate(thres, thres, element, dil2);
            cvErode(thres2, thres2, element, ero1);
            cvDilate(thres2, thres2, element, dil1);
            cvErode(thres2, thres2, element, ero2);
            cvDilate(thres2, thres2, element, dil2);
            //==================PROSES THRESHOLDING
DAN
PENENTUAN=============================================
=====
            for (int x = 0;x<hasil->width;x = x++)
                  for (int y = 0;y<hasil->height;y =
y++)
                  {if (thres->imageData[thres-
>widthStep*y + x*thres->nChannels] == 0)
```

```
                                    hasil-
                                    >imageData[hasil-
                                    >widthStep*y+x*hasil>
                                    nChannels] = 255;
                              else
                                    hasil-
                                    >imageData[hasil-
                                    >widthStep*y+x*hasil>
                                    nChannels] = 0;
                        }
                for (int x1 = 0;x1<hasil2->width;x1 =
x1++)
                        for (int y1 = 0;y1<hasil2-
>height;y1 = y1++)
                        {
                if (thres2->imageData[thres2->widthStep*y1
+ x1*thres2->nChannels] == 0)
                hasil2->imageData[hasil2->widthStep*y1 +
x1*hasil2->nChannels] = 255;
                else
                hasil2->imageData[hasil2->widthStep*y1 +
x1*hasil2->nChannels] = 0;
                        }
//==========MENDETEKSI OBJEK==================//
        for (int x = 0;x<thres->width;x = x++)
        for (int y = 0;y<thres->height;y = y++)
        {if (hasil->imageData[hasil->widthStep*y +
x*hasil->nChannels] == 0)
                                {ukur.x = x;}}
                for (int x = 0;x<thres2->width;x = x++)
                for (int y = 0;y<thres2->height;y = y++)
                        {if (hasil2->imageData[hasil2-
>widthStep*y+x*hasil2->nChannels]== 0)
                                {ukur2.x = x;}}
                for (int y = 0;y<thres->height;y = y++)
                for (int x = 0;x<thres->width;x = x++)
        {if(hasil->imageData[hasil->widthStep*y+x*hasil-
>nChannels]== 0)
                                {ukur.y = y;}}
                for (int y = 0;y<thres2->height;y = y++)
```

```
              for (int x = 0;x<thres2->width;x = x++)
{if (hasil2->imageData[hasil2->widthStep*+ x*hasil2-
>nChannels] == 0)
                              {ukur2.y = y;}}
for (int x = thres->width - 1;0<x;x = x--)
       for (int y = thres->height - 1;0<y;y = y--)
{if (hasil->imageData[hasil->widthStep*y+x*hasil-
>nChannels] == 0)
              {ukur1.x = x;}}
       for (int x = thres2->width - 1;0<x;x = x--)
       for (int y = thres2->height - 1;0<y;y = y--)
       {if (hasil2->imageData[hasil2-
>widthStep*y+x*hasil2->nChannels] == 0)
                              {ukur3.x = x;}}
       for (int y = thres->height - 1;0<y;y = y--)
       for (int x = thres->width - 1;0<x;x = x--)
{if (hasil->imageData[hasil->widthStep*y + x*hasil-
>nChannels] == 0)
                              {ukur1.y = y;}}
       for (int y = thres2->height - 1;0<y;y = y--)
       for (int x = thres2->width - 1;0<x;x = x--)
                      {if (hasil2->imageData[hasil2-
>widthStep*+ x*hasil2->nChannels] == 0)
                              {ukur3.y = y;}}
//==============ngotaki disekeliling objek==========
              cvRectangle(frame, cvPoint(ukur.x,
ukur.y), cvPoint(ukur1.x, ukur1.y), cvScalar(0, 0, 255,
0), 2, 0, 0);
              cvRectangle(frame2, cvPoint(ukur2.x,
ukur2.y), cvPoint(ukur3.x, ukur3.y), cvScalar(0, 0,
255, 0), 2, 0, 0);
//==========MENCARI NILAI X,Y, dan Z==============//
int xfix = ((ukur.x - ukur1.x) / 2) + ukur1.x;
int yfix = ((ukur.y - ukur1.y) / 2) + ukur1.y;
int xfix2 = (((ukur2.x - ukur3.x) / 2) + ukur3.x);
int yfix2 = (((ukur2.y - ukur3.y) / 2) + ukur3.y);
int f = 6.3445298876926358e+002;
int d = ((640 - xfix2) + xfix);;
int s = abs(300 * f) / d;
int z = s + ((0.028*(s*s)) - (12.963*s) + 1388.4);
```

```
                int t = (xfix2*z) / f;
                if (int(z >= 140) && (z<180))
                {x = t
                }else if ((z >= 180) && (z<220))
                {x = t + (-1.4767*t) + 15.37;}
                else if ((z >= 220) && (z<260))
                {x = t + (-1.4715*t) + 25.894;
                }else if ((z >= 260) && (z<270))
                {x = t + (-1.5032*t) + 33.438;}
                else if ((z >= 270) && (z<310))
                {x = t + (-1.5149*t) + 30.429;}
                int u = (yfix2*z) / f;
                if (int(z >= 80) && (z<140))
                {y = u + (-1.6025*u) + 18.364;}
                else if ((z >= 140) && (z<180))
                {y = u + (-1.5356*u) + 28.667;
                }else if ((z >= 180) && (z<220))
                {y = u + (-1.4995*u) + 32.557;}
                else if ((z >= 220) && (z<260))
                {y = u + ((-1.4898*u) + 40.223);}
                else if ((z >= 260) && (z<290))
                {y = u + (-1.5111*u) + 46.103;}
                else if ((z >= 290) && (z<310))
                {y = u + ((-1.4949*u) + 47.189);}
                xs = (1 * x) + (0 * y) + (0 * z) + 0;
                ys = (0 * x) + (1 * y) + (0 * z) - 16;
//sebelumnya 16 sebagai jarak antara kamera dengan
pusat senapan
printf("x senjata= %.2f\t ysenjata= %.2f\n", xs, ys);
teta1 = 90-(atan((double)xs / (double)z) * (180 / PI));
teta2 = 90-(atan((double)ys / (double)z) * (180 / PI));
       float predX = 1, predY = 1, predZ = 1, nextX = 1,
nextY = 1, nextZ = 1, prevX = 1, prevY = 1, prevZ = 1;
       float alpha = 0.1;
predX = predX + alpha*(xs - predX);prevX = predX;
predY = predY + alpha*(ys - predY);prevY = predY;
predZ = predZ + alpha*(z - predZ);prevZ = predZ;
printf("x== %.2f  \t y== %.2f  \t z== %.2f\n", predX,
predY, predZ);
                cvShowImage("hasil", hasil);
```

```
                cvShowImage("left result", frame);
                cvShowImage("hasil2", hasil2);
                cvShowImage("right result", frame2);
                char c = cvWaitKey(99);
                if (c == 'c') {
                        mousev.clear();
                        kalmanv.clear();
                        cvZero(gbKalman);
                }
                if (cvWaitKey(1) == 'q')
                        break;
                cvReleaseImage(&thres);
                cvReleaseImage(&thres2);
                cvReleaseImage(&HSV);
                cvReleaseImage(&HSV2);
                cvReleaseImage(&hasil);
                cvReleaseImage(&hasil2);
        }
        cvDestroyWindow("Filter");
        cvDestroyWindow("Color");
        cvReleaseImage(&frame);
        cvReleaseImage(&frame2);
        cvReleaseImage(&gbKalman);
        cvReleaseCapture(&capture);
        cvReleaseCapture(&capture2);
        cvDestroyWindow("left result");
        cvDestroyWindow("right result");
        fclose(fh);
        return 0;
}
float mapping(float x, float in_min, float in_max,
float out_min, float out_max)
{return (x - in_min) * (out_max - out_min) / (in_max -
in_min) + out_min;}
```

BAB VI
KESIMPULAN DAN SARAN

6.1 Kesimpulan

Adapun kesimpulan dari penelitian tugas akhir ini adalah

a. Prediksi koordinat 3 dimensi dengan metode *exponential smoothing* telah didapat dengan alfa 0,4995.
b. Selisih ERMS koordinat x dan y dari 10 video, 6 diantaranya menunjukkan bahwa ERMS *exponential smoothing* mempunyai nilai lebih kecil disbanding filter kalman. Hasil nilai error terbesar pada metode filter kalman untuk koordinat x adalah 8.55cm dan untuk koordinat y sebesar 18.34cm, sedangkan hasil error terbesar metode *exponential smoothing* untuk koordinat x adalah 11.126cm dan untuk koordinat y sebesar 12.77cm.

6.2 Saran

Berdasarkan analisa *error* dan pembahasan dalam tugas akhir ini, terdapat beberapa saran terkait pengembangan penentuan prediksi koordinat 3 dimensi suatu objek yaitu :

a. Kamera yang digunakan sebisa mungkin menggunakan fps diatas 30, agar data yang ditangkap lebih banyak dan dengan resolusi yang lebih tinggi.
b. Pengambilan video dengan jarak yang berbeda-beda agar dapat diketahui hubungan antara jarak pengambilan dengan data yang diperoleh.
c. Pada pengambilan data, jarak bola ke kamera adalah 160cm dan stereo kamera dengan jarak antar kamera 30cm hanya dapat menangkap frame selebar 30cm. Akan lebih baik jika jarak antar kamera dikurangi, atau kamera menghadap kedalam sehingga area yang ditangkap menjadi lebih luas.
d. Perlu diteliti lebih lanjut tentang pengaruh lintasan bola terhadap prediksi.

(Halaman ini sengaja dikosongkan)

Daftar Pustaka

[1] Kiki Prawiroredjo., Nyssa Asteria. *Detektor jarak Dengan Sensor Ultrasonik Berbasis Mikrokontroler*. Jakarta :Universitas Trisakti; 2008.

[2] Angelo Gagliardi., Dante Gagliardi. *Autonomous/Remote Pilot Sentry Gun Platform*. California: California Polytechnic State University; 2013.

[3] Wardah Choirina Lutfi. *Kalibrasi Strereo Kamera dan Penentuan Koordinat 3 Dimensi untuk Target Tunggal pada Sistem Pelontar Peluru Autotracking*. Surabaya: Institut Teknologi Sepuluh Nopember; 2016.

[4] Wahyudi A, Sasongko Pramono H, Wahyu Widada, *Simulasi Filter Kalman untuk Estimasi Posisi dengan Menggunakan Sensor Accelerometer.* Semarang: Universitas Diponegoro; 2007.

[5] Marco André F. *Intelligent Tracking of Handball Players at F.C.PORTO and FADEUP*. Spanyol: Universidade do Porto; 2011.

[6] Fifi Puspita Ningsih. *Pengukuran Kecepatan Objek Bergerak Menggunakan Webcam Berbasis Pengolahan Citra Digital*. Surabaya: Institut Teknologi Sepuluh Nopember; 2011.

[7] Kenneth Haugaard Møller. *3D Object Modelling via Registration of Stereo Range Data*. Denmark: University of Denmark; 2006.

[8] Erik G. Learned-Miller. *Introduction to Computer Vision*. Amherst: University of Massachusetts; 2011.

[9] Tri Arief Sardjono, Djoko Purwanto dan Achmad Fiqhi Ibadillah.*Sistem Penjejakan Obyek Dengan Stereo Vision*. Surabaya: Institut Teknologi Sepuluh Nopember; 2011.

[10] Anggit Praharasty. *Simulasi Pendeteksian Garis Lurus (Straight Line) Pada Citra Digital Menggunakan Matlab 7.1*. Surabaya: Institut Teknologi Sepuluh Nopember; 2008.

[11] Dhini Wafiroh. *Rancang Bangun Program Deteksi Plat Nomor dan Pengenalan Alfanumerik Menggunakan Metode*

SVM dan K-Nearest Neighbor. Surabaya: Institut Teknologi Sepuluh Nopember; 2015.

[12] Mohd Nazri Abu Bakar., Abdul Rahman Mohd. Saad. *A Monocular Vision-based Specific Person Detection System for Mobile Robot Applications*. Malaysia: Universiti Malaysia Perlis; 2012.

[13] Gary Bradski and Adrian Kaehler. *Learning OpenCV*. Amerika: O'Reilley; 2008.

[14] Richard B. Chase., F. Robert Jacobs. "*Operation Management for Competitive Advantage 11th edition*". New York: McGraw-Hill; 2006.

[15] Greg Welch., Gary Bishop. *An Introduction to the Kalman Filter*. Chapel Hill: University of North Carolina; 2001.

[16] Julius Oyeleke. *Kalman Tracking for Image Processing Application.* Irlandia: OĖ Gaillimh – NUI Galway; 2010.

BIODATA PENULIS

Valya Ika Dhanie lahir di kota kecil Kediri, pada tanggal 3 September 1994. Dari pasanagan Alm. Heru Prasetyo dan juga Endah Retno Asih. Penulis menghabiskan masa kecilnya dikota kelahiran dari TK Al-Irsyad, SDN Singonegaran 1 yang dekat dengan rumah nenek sehingga hanya perlu berjalan kaki jika pulang, lalu SMP Negeri 1 Kediri dan menempuh SMA di SMA Negeri 2 Kota Kediri.

Penulis baru hijrah ke kota Surabaya setelah diterima kuliah pada jurusan Teknik Mesin ITS lewat jalur undangan. Pada saat kuliah penulis pernah menjadi staff pada BEM fakultas, lalu juga aktif dalam berbagai kegiatan yang diadakan himpunan mesin sendiri. Mengambil bidang manufaktur dan memilih laboratorium Perancangan dan Pengembangan Produk sebgai rumah kedua. Hobi dari penulis adalah membaca, mulai dari membaca novel, biografi, dan juga komik. Penulis juga suka menulis cerita pendek juga menggambar tokoh animasi. Menyukai musik lebih dari apapun mulai dari genre Rock Alternatif hingga K-pop, namun tidak menyukai musik dangdut.

Jika ada informasi atau apapun yang ingin disampaikan kepada penulis bisa menghubungi lewat email dvalyaika@yahoo.co.id.